IBM® WebSphere®

WebSphere Application Server Liberty Profile Guide for Developers

- Learn WebSphere Application Server profile designed for developers
- Lightweight, easy to install, fast to use application server
- Simplified development with the Liberty profile

Brent Daniel
Erin Schnabel
Kevin Smith

Redbooks

ibm.com/redbooks

International Technical Support Organization

WebSphere Application Server Liberty Profile: Guide for Developers

November 2012

SG24-8076-00

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (November 2012)

This edition applies to WebSphere Application Server V8.5 Liberty Profile.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®	Passport Advantage®	Tivoli®
DB2®	Rational®	WebSphere®
IBM®	Redbooks®	z/OS®
MVS™	Redbooks (logo)  ®	

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Preface

WebSphere® Application Server V8.5 includes a Liberty profile, which is a highly composable, dynamic application server profile. It is designed for two specific use cases: Developer with a smaller production runtime, and production environments. For a developer, it focuses on the tasks a developer does most frequently and makes it possible for the developer to complete those tasks as quickly and as simply as possible. For production environments, it provides a dynamic, small footprint runtime to be able to maximize system resources.

This IBM® Redbooks® publication provides you with information to effectively use the WebSphere Application Server V8.5 Liberty profile along with the WebSphere Application Server Developer Tools for Eclispe, for development and testing of web applications that do not require a full Java Platform. It provides a quick guide on getting started, providing a scenario-based approach to demonstrate the capabilities of the Liberty profile along with the developer tools, providing a simplified, but comprehensive, application development and testing environment.

The intended audience for this book is developers of web and OSGi applications who are familiar with web and OSGi application concepts.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Brent Daniel is a developer for WebSphere Application Server Liberty profile. He has worked in the WebSphere organization for over 13 years on a diverse set of technologies including persistence, web applications, administration, and SCA.

Erin Schnabel is the Development lead for the WebSphere Application Server Liberty profile. She has over 12 years of experience in the WebSphere Application Server development organization in a variety of technical roles. Erin has over 15 years of experience working with Java and application middleware across a variety of hardware platforms, including z/OS®. She specializes in composable runtimes, including the application of OSGi, object-oriented and service-oriented technologies and design patterns to decompose existing software systems into flexible, composable units.

Kevin Smith is the test architect for the Websphere Application Server Liberty profile. Based in Hursley in the United Kingdom he has worked in IBM for almost 20 years on all aspects of the development lifecycle. For the last five years Kevin has been focused on innovating the way we test in an Agile development environment and most recently has been applying these practices to the Liberty profile to drive increased quality and efficiency across the worldwide development teams.

Thanks to the following people for their contributions to this project:

Margaret Ticknor, Deana Coble, Shari Deiana, Tamikia Lee, Linda Robinson
International Technical Support Organization, Raleigh Center

Alfred Schwab
International Technical Support Organization, Poughkeepsie Center

Alex Mulholland, Gary R. Picher, Nathan Rauh
IBM US

Ian Robinson
IBM UK

Jacek Laskowski
IBM Poland

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

An introduction to the Liberty profile

The IBM WebSphere Application Server V8.5 Liberty Profile is a composable, dynamic application server environment that supports development and testing of web applications that do not require a full Java Platform, Enterprise Edition 6 (Java EE 6) stack. Though the profile is fully supported for production use, it is designed to work alongside WebSphere Application Server Developer Tools for Eclipse. Working with the developer tools, it provides a simplified, but comprehensive, application development and testing environment.

This guide is for developers of web and OSGi applications. It assumes familiarity with web and OSGi application concepts, but does not assume familiarity with WebSphere Application Server resources or products. This guide refers to other IBM Redbooks publications and the WebSphere Application Server V8.5 Information Center for additional information.

In this chapter, we present an overview of the Liberty profile emphasizing characteristics that are especially useful for iterative and collaborative application development. This overview chapter covers the following topics:

- Liberty profile overview
- Simplified configuration
- Runtime composition with features and services
- Frictionless application development

1.1 Liberty profile overview

The Liberty profile is a simplified, lightweight development and application runtime environment that is:

- Simple to configure. Configuration is read from a single xml file with text-editor-friendly syntax.
- Dynamic and flexible. The runtime loads only what your application needs and recomposes the runtime in response to configuration changes.
- Fast. The server starts in under five seconds with a basic web application.

The Liberty profile is built using OSGi technology and concepts. The fit-for-purpose nature of the runtime relies on the dynamic behavior inherent in the OSGi Framework and Service Registry. As bundles are installed to or uninstalled from the framework, the services each bundle provides are added or removed from the service registry. The addition and removal of services similarly cascades to other dependent services. The result is a dynamic, composable runtime that can be provisioned with only what your application requires and responds dynamically to configuration changes as your application evolves.

1.1.1 Programming models

The Liberty profile supports a subset of the Java EE 6 stack. The following list notes the supported technologies:

- Java Servlet 3.0
- JavaServer Faces (JSF) 2.0
- JavaServer Pages (JSP) 2.2
- Java Expression Language 2.2
- Standard Tag Library for JavaServer Pages (JSTL) 1.2
- Bean Validation 1.0
- Java Persistence API (JPA) 2.0
- Java Transaction API (JTA) 1.1
- Java API for RESTful Web Services (JAX-RS) 1.1

The Liberty profile also supports OSGi applications. The following list shows supported technologies for OSGi applications (with a reference to the specification where appropriate):

- Web Application Bundles (OSGi R4.2 Enterprise, Chapter 128)
- Blueprint Container (OSGi R4.2 Enterprise, Chapter 121)
  - Blueprint Transactions
  - Blueprint Managed JPA
- JNDI (OSGi R4.2 Enterprise, Chapter 126)

- OSGi application of Java EE technologies supported by the profile

A complete list of technologies supported by the Liberty profile can be found at the following information center website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.doc/topics/rwlp_prog_model_support.html

1.1.2 Supported development environments

Both Liberty and full profile WebSphere Application Server support a broad list of operating systems: AIX®, HP, IBM i, Linux, Solaris, Windows, and z/OS. The Liberty profile is also supported, for development, on Mac OSX.

The list of supported operating systems for the Liberty profile can be found at:

http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg27028175

Requirements for running the Liberty profile on z/OS can be found at:

http://www-01.ibm.com/support/docview.wss?uid=swg27024798

The Liberty profile server will run on Java version 6 or above provided by any vendor. A full list of supported Java versions is noted in the system requirements documents at:

http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg27028175

1.1.3 Additional resources

There are additional resources available to support development with the Liberty profile. The following list notes developer-community focused resources:

- WASdev community

  http://wasdev.net/

  The WASdev community is a hub for information about developing applications for WebSphere Application Server, and using the Liberty profile in particular. Articles, podcasts, videos, and samples are refreshed regularly as new technology is made available through early access programs or new product releases.

- WASdev forum

  http://www.ibm.com/developerworks/forums/forum.jspa?forumID=2666

  The WASdev forum provides an opportunity for users of WebSphere Application Server and the Liberty profile to interact with each other and with the IBM developers working on the products.

- Stack Overflow tags

  http://stackoverflow.com/questions/tagged/websphere-liberty

  Stack Overflow is a Q&A site that has no cost to use. It allows users to ask and answer questions with built-in incentives to reward high quality answers. Use the `websphere` or `websphere-liberty` tags to ask or answer questions about WebSphere Application Server and the Liberty profile.

1.2 Simplified configuration

The Liberty profile runtime environment is composed of a set of OSGi bundles. Any bundle containing configurable resources also contains metadata describing the configuration those resources accept or require using the OSGi Metatype service. A generic description of a metatype service is a bundle containing within it an XML schema describing each resource's configuration attributes, types, and value bounds. The configuration metadata within each bundle will also assign appropriate default values, and can construct default instances where appropriate. With functional defaults already in place, you only need to specify the values you want to customize.

1.2.1 Server definition

The Liberty profile supports multiple servers using the same installation. Each server is defined in its own directory, where the directory name is the server name. New servers are easily created using the command line or the developer tools, but manual creation will also work. Using the developer tools to create a server is discussed in Section 2.2, "Install the Liberty profile" on page 26.

A server definition requires only one configuration file, such as the `server.xml` file, to define a server. The `server.xml` file must contain a server element. If there are no configuration changes then default values suffice. The server element can be empty, as shown in Example 1-1.

Example 1-1 Server.xml file

```
<server description="optional description" />
```

The empty definition in Example 1-1 only launches the core kernel. Additional function is enabled using features. Features are discussed in 1.3.1, "Feature management" on page 12.

1.2.2 Server configuration using the server.xml file

The `server.xml` file has a simple XML format that can be edited using your favorite text editor. Though there is no requirement to use a GUI to edit your server's configuration, developer tools provides an enhanced editor. This editor has context-sensitive help, wizard-style value selection, and built-in configuration validation. The enhanced editor also allows direct viewing and editing of the XML source.

Figure 1-1 shows the design and source tabs of the enhanced server.xml editor in the developer tools.

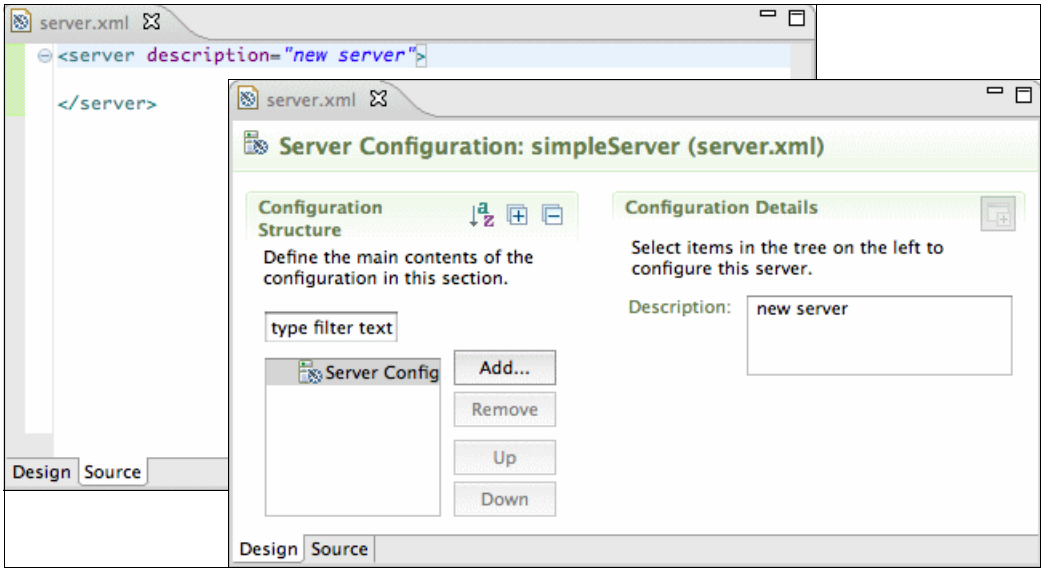


Figure 1-1 The source and design tabs show different views of the server's configuration

The subsequent chapters in this publication provide examples using the enhanced editor to make server configuration updates. If you follow the examples, you might find it interesting to switch to the source view. There you can observe changes in the `server.xml` file after using the enhanced editor to modify the server's configuration.

1.2.3 Bootstrap properties

There is another configuration file, used by the Liberty profile server when initializing the OSGi framework and platform core, called `bootstrap.properties`. The `bootstrap.properties` file is an optional peer of the `server.xml` file containing simple text properties as `key=value` pairs. Bootstrap properties are

generally used to influence early server start behavior or to define machine-specific properties.

1.2.4 Portable configuration using variables

The Liberty profile configuration manager resolves variables when processing configuration information. Using configuration variables allows environment-specific attributes to be removed from the bulk of the server's configuration. This isolates general aspects of server or application configuration from environment-specific elements such as hostnames, port numbers, or file system locations.

The Liberty profile provides several predefined variables to help make server configurations portable. The following list demonstrates syntax and briefly summarizes the supported location variables:

`wlp.install.dir` This is the profile's installation directory.

`wlp.user.dir` This directory is the root of the tree containing user configuration. By default, this directory is a child of the install directory, `${wlp.install.dir}/usr`. It can be configured to an alternate location using an environment variable when the server is started.

`server.config.dir` The server configuration directory contains the `server.xml` file and any other optional configuration files used to customize the server's environment. The name of this directory is used as the server's name. By default, this directory is a child of the user directory called `${wlp.user.dir}/servers/`*`serverName`*.

`server.output.dir` The server output directory contains resources generated by the server, such as logs. The Liberty profile supports shared configurations, for example, multiple running server instances sharing the same configuration files. Shared configurations are supported by allowing the instance-specific output directory to be moved using an environment variable when the server is started. By default, the server output directory is the same as the server's configuration directory.

`shared.app.dir` Shared user applications directory, `${wlp.user.dir}/shared/apps`.

`shared.config.dir` Shared user configuration directory, `${wlp.user.dir}/shared/config`.

`shared.resource.dir` Shared user resources directory, `${wlp.user.dir}/shared/resources`.

When specifying file or resource locations, use the most specific location variable that applies. Being specific helps to ensure that your server configuration can be propagated to other environments (a fellow developer's notebook, a dedicated test automation framework, a distributed production environment) without requiring changes to the `server.xml` file.

Example 1-2 shows a simple `bootstrap.properties` file that defines a custom variable. It is defined using a built-in variable and additional variables for machine-specific host and ports.

Example 1-2 A simple bootstrap.properties file

```
appCacheFile=${server.output.dir}/application/cache
hostName=*
httpPort=9082
httpsPort=9445
```

Figure 1-2 provides an example of a `server.xml` file that references the variables from the `bootstrap.properties` file from Example 1-2.

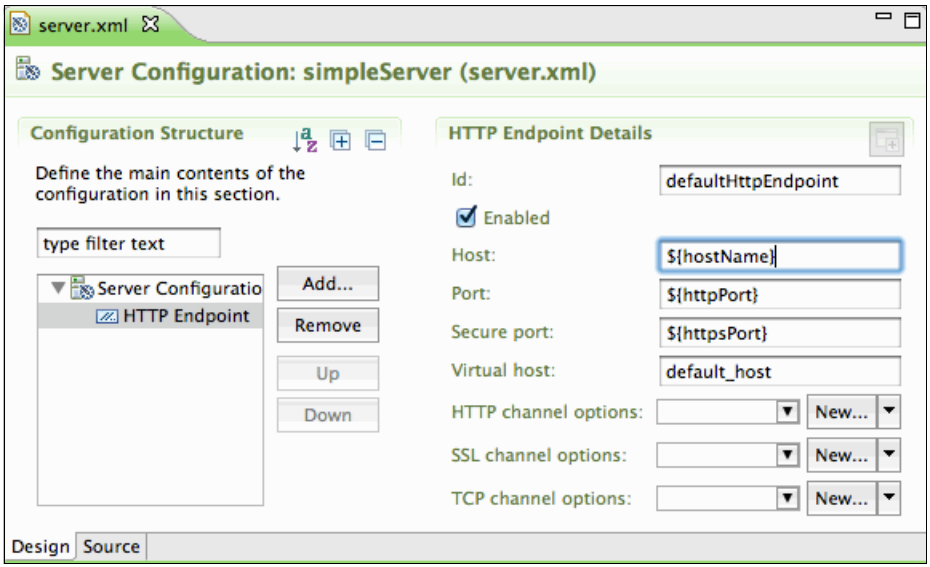


Figure 1-2 Using variables in the enhanced editor

The enhanced editor provides content assist for variables defined in either the `bootstrap.properties` or `server.xml` files. Variables are filtered based on the

field being configured. If the field is for a number (integer, long, and so on), the content assist list will only show variables with numeric values. This context-sensitive variable filtering is shown in Figure 1-3.

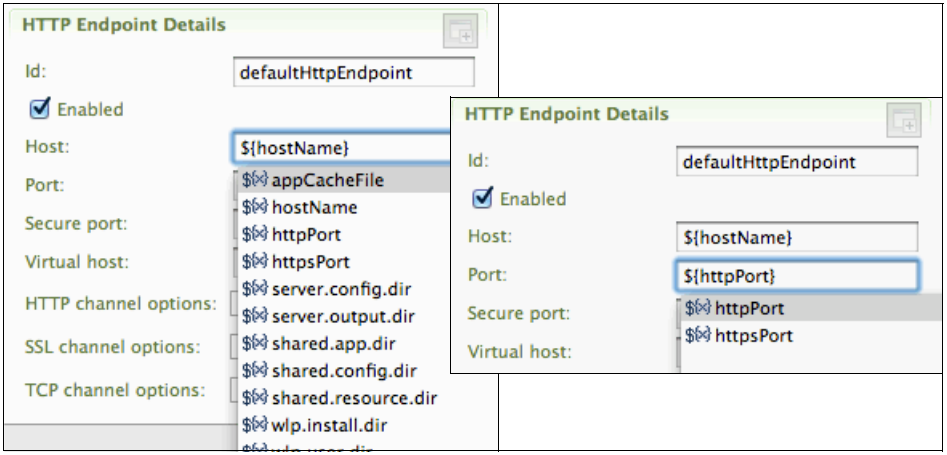


Figure 1-3 Content assist will filter-defined variables

1.2.5 New configuration types and validation

The simple XML format used in `server.xml` uses intuitive, natural types to make configuring the server more natural. For example, durations and timeouts can be specified using strings containing units (specify 1m30s for 1 minute and 30 seconds).

The server is tolerant of configuration mistakes where possible. If the initial configuration for the server is malformed, the server will not start, but will indicate where the syntax error occurred to help with correcting the problem. If the configuration for a running server is updated with invalid syntax, the change is not applied, and the server continues to run. It will also display a message indicating where the configuration problem was so it can be corrected. Subsequent updates to correct the error will be applied to the server without requiring a server restart. Configuration provided for unknown or inactive elements is ignored: you do not have to remove configuration for features that have been disabled.

The following information center website provides an index describing all configurable attributes of the runtime:

`http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.doc/autodita/rwlp_metatype_4ic.html`

1.2.6 Encoding passwords

Passwords can be specified as encoded strings to alleviate concerns about using plain text passwords in configuration files. There are two ways to create and specify an encoded password:

- The command line. You can use the **securityUtility** script in the `${wlp.install.dir}/bin` directory:

  ```
  securityUtility encode myPassword
  ```

  This will return an encoded string, starting with **"**`{xor}`**..."**, to be copied verbatim into your `server.xml` file.

- The enhanced editor. When the editor is used to configure passwords, a standard password dialog box performs the same encoding step and sets the attribute value to the resulting encoded string.

1.2.7 Shared configuration using includes

The Liberty profile also supports composed configuration. It is possible for the `server.xml` file to include other files. Included files are specified using the `<include.../>` directive, and must use the `server.xml` file's syntax. Each included file must also contain a `<server.../>` element. Figure 1-4 on page 10 provides a basic representation of how the configuration manager works with configuration coming from various sources. Each bundle in the runtime provides configuration defaults and configuration metadata. The configuration manager reads this information from each bundle and merges it with information read from the `server.xml` file and any included files. The configuration manager will stop processing, with an error, if any required configuration files cannot be found.

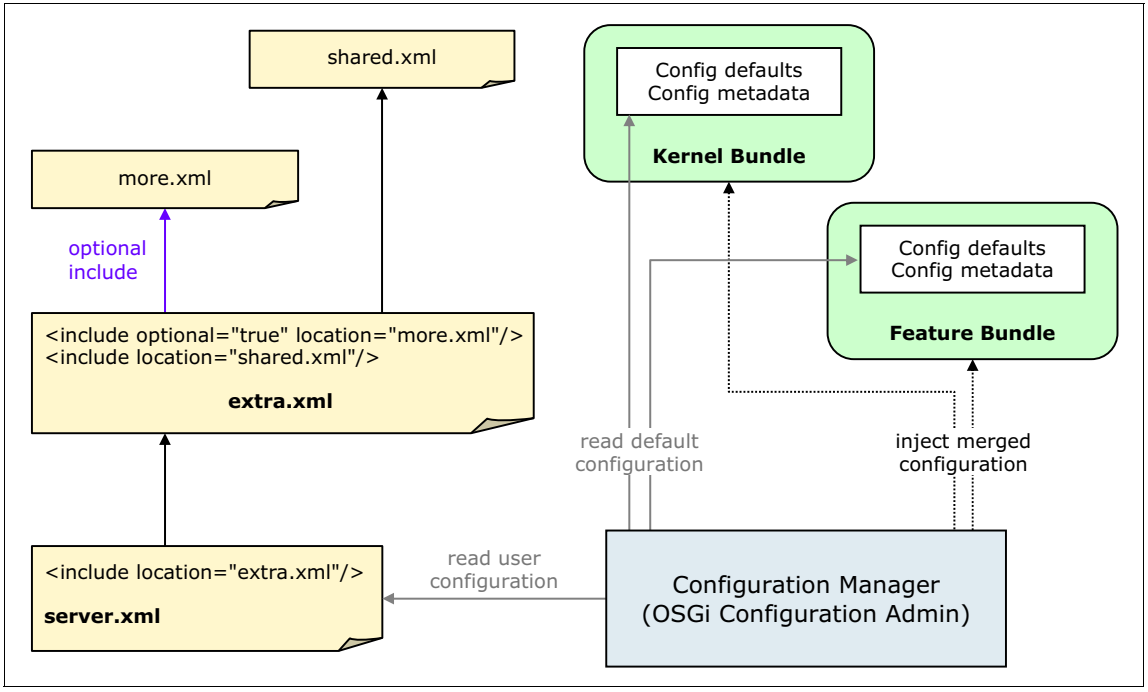


Figure 1-4 The configuration manager processing includes configuration

Through the use of variables and included files, a wide variety of configuration patterns becomes possible. Configuration can be shared across servers or persisted in a source control repository at varying levels of granularity. For example, all configurations associated with a data source can be collected into a single xml file that is reused (as an included resource) across multiple server configurations. The data source configuration can use variables (for output directories, hostnames, ports, and so on) so that the data source configuration can be applied directly across all servers unchanged.

Included files can be specified using full paths, paths with variables, or relative paths. Relative paths are resolved by looking in a location relative to the parent file, the server's configuration directory, or the shared configuration directory (in that order).

Include statements can appear anywhere within the `<server.../>` configuration element, and each included file can include other files. To ensure that your included configuration behaves predictably, you need to be aware of the following processing rules for included configuration files:

- Each included file is logically merged into the main configuration at the position that the `<include />` statement occurs in the parent file.

- For configuration instances such as applications or endpoints, a unique configuration is created for each appearance of the element in the xml file. An ID is generated if not otherwise specified.
- For singleton services, such as logging or application monitoring, entries are processed in the order they appear in the file and later entries add to or override previous ones. This is also true for configuration instances (an application or data source) where the configuration instances have the same ID.

1.2.8 Dynamic configuration updates

The Liberty profile contains an implementation of the OSGi Configuration Admin service, which operates using the service patterns defined in that specification. Per that specification, the Liberty profile configuration manager does the following:

- Maintains and manages the configuration of the OSGi Service Platform as a whole. It sets the active configuration, at runtime, for all configurable services.
- Monitors the OSGi Service Registry and provides configuration information to configurable services as they are registered.

At startup, the configuration manager processes the contents of the `server.xml` file. It then merges those contents with the default values provided by each bundle to provide configuration information to each registered configurable service. If the `server.xml` file changes, the configuration manager will reprocess the file contents and immediately push configuration changes out to the configurable services affected by the change. There is no need to restart the server.

The configuration manager also responds dynamically as bundles are added or removed from the system. For example, a new bundle might introduce additional configurable services. When a bundle is started, the configuration manager:

- Detects new services.
- Resolves any user-provided configuration against the new configuration metadata and defaults provided by the new bundle.
- Provides the resolved configuration to the consuming services.

Because the runtime is dynamic, the configuration manager is tolerant when it processes configuration information from any source (user or system). The configuration manager will safely ignore configuration elements or attributes for which it does not have metadata. In the case of unknown attributes, the values are passed to the associated configurable services along with the rest of its configuration data. For unknown configuration definitions, the configuration

manager parses configuration information at the time it becomes available and provides it to associated configurable services.

1.3 Runtime composition with features and services

The composable nature of the Liberty profile is based on the concept of features. A feature is a unit of function, essentially the minimum set of bundles required to support a particular technology. Features can and do overlap, and they can include other features.

As an example, the `servlet-3.0` feature installs the collection of OSGi bundles required to support Servlet applications, but not JSP applications. The `jsp-2.2` feature includes the `servlet-3.0` feature (it requires a servlet engine). When you enable the `jsp-2.2` feature in a server, it installs all of the bundles required for servlet applications in addition to the bundles required to support JSPs (see Figure 1-5).

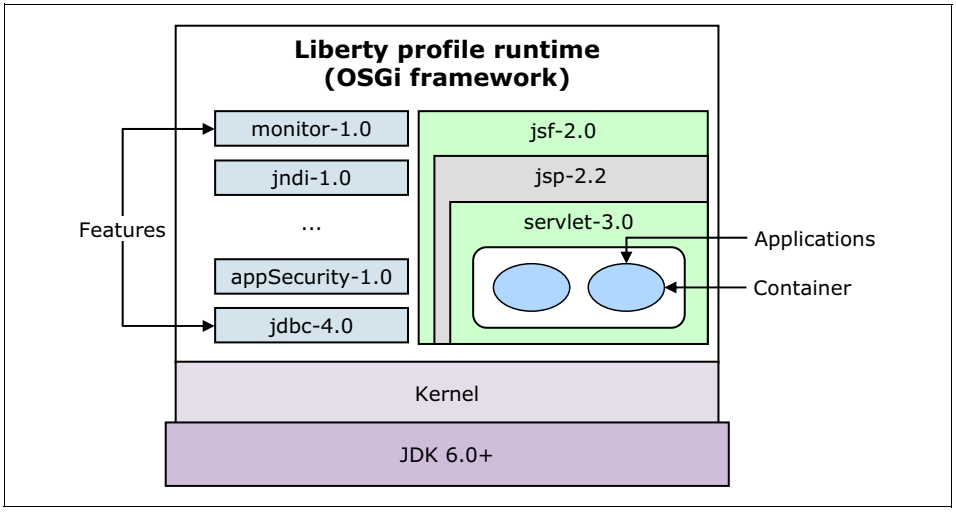


Figure 1-5 Features are the units of composition for a Liberty profile server runtime

1.3.1 Feature management

Features are configured in `server.xml` by adding `<feature>` elements to the `<featureManager../>`. When a new server is created using the **`server create`** command, the `server.xml` file has the `jsp-2.2` feature enabled, as shown in Example 1-3 on page 13.

Example 1-3 Server.xml file with jsp-2.2 feature

```
<featureManager>
   <feature>jsp-2.2</feature>
</featureManager>
```

The developer tools enhanced editor provides a list of features to choose from and shows those features implicitly enabled by others, as shown in Figure 1-6.

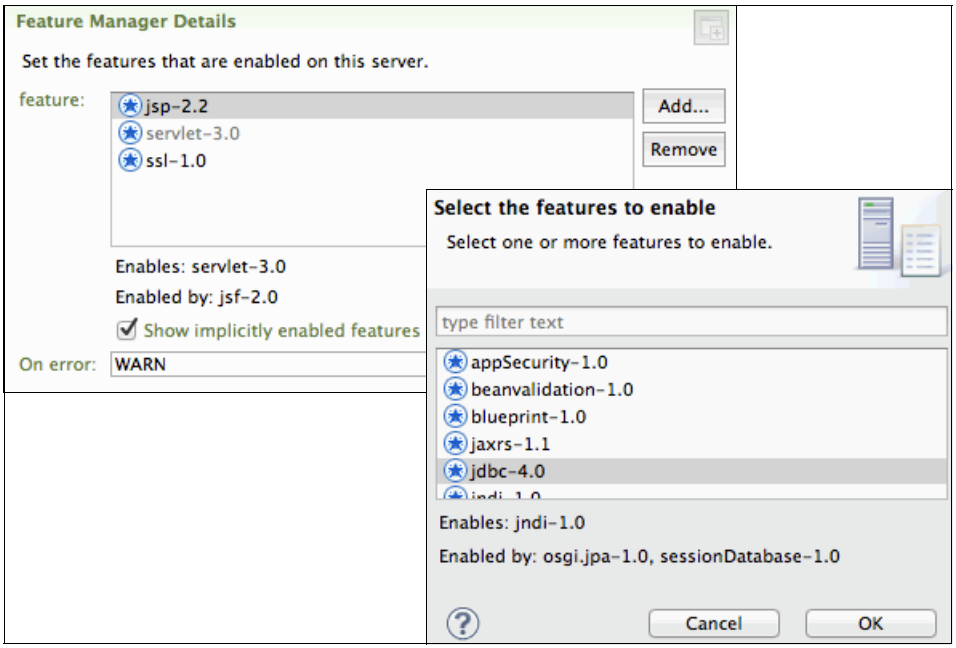


Figure 1-6 Selecting features with the enhanced editor

Features can be added or removed from the server configuration while the server is running without requiring a server restart.

1.3.2 Automatic service discovery

The fit-to-purpose nature of the runtime relies on dynamic behavior inherent in the OSGi Framework and the Service Registry. The Liberty profile relies on the OSGi Service Registry for dynamic registration and discovery of services.

When a feature is enabled, the bundles required by that feature are installed in the OSGi framework. When a bundle is installed and started, in some cases, services are registered in the Service Registry. For most bundles in the Liberty

profile, service registration and lifecycle are managed using OSGi Declarative Services (DS) components.

DS support operates on declared components, each of which is defined by an XML file in the bundle. When a bundle containing component declarations is added to the framework, DS takes action. It reads each component declaration and registers provided services in the service registry. DS then manages the lifecycle of the component: controlling its lifecycle based on a combination of declared attributes and satisfied dependencies.

The simple XML description of components allows DS to resolve service dependencies without requiring the component to be instantiated, or its implementation classes to be loaded. All of this facilitates late and lazy resource loading, which helps improve server startup and runtime memory footprint.

1.4 Frictionless application development

The Liberty profile can deploy applications in the following ways:

- As an archive file (WAR, WAB, EAR, or EBA)
- Extracted into a directory
- Described by an xml file (loose configuration)

There is no distinction between installing and starting an application, though installed applications can be stopped and restarted.

By default, the Liberty profile will monitor deployed applications for changes. Updates to static files (html, css, or javascript) or JSP files are detected and served immediately. Changes to servlet classes cause an automatic restart of the application.

Caution: For large applications, monitoring the file system for changes to an application can be resource intensive. Consider extending the monitoring interval if you see excessive I/O activity while the server is running. Application update monitoring should be disabled in production environments. For information, see 8.1.1, “Turn off application monitoring” on page 140.

1.4.1 Quick start using dropins

By default, each server contains a monitored application directory named `dropins`. When an application is placed in this directory, the server will automatically deploy and start the application.

There are several options for placing applications in the `dropins` directory. Each provides a way for the server to determine the application type. The following list describes the placement options:

- Place the archive file with its identifying suffix (ear, war, wab, and so on) directly into the `dropins` directory:

  `${server.config.dir}/dropins/myApp.war`

- Extract the archive file into a directory named with the application name and the identifying suffix:

  `${server.config.dir}/dropins/myApp.war/WEB-INF/...`

- Place the archive file or the extracted archive into a subdirectory named with the identifying suffix:

  `${server.config.dir}/dropins/war/myApp/WEB-INF/...`

1.4.2 Configuration-based application deployment

Explicitly declaring your application allows you to specify additional properties such as the context root, alter application startup behavior, and so on. Explicitly configured applications are recommended for production environments, to allow dropins monitoring to be disabled.

An application is declared in the `server.xml` file using the `<application.../>` element, which supports the following attributes:

location	The path to the application resource
id	A unique identifier for the application
name	A descriptive/display name for the application
type	The application type: WAR, EAR, EBA. If this is not specified, it will be inferred (if possible) from the application file name.

The location is the only required attribute for a declared application. It specifies the location of the application resource (archive, extracted directory, or loose configuration xml file) and can refer to any location in the file system. If the specified location is a relative path, it will be resolved first against `${server.config.dir}/apps` and then against `${shared.app.dir}`.

Update monitoring for declared applications has a configurable trigger. By default, the file system is polled for changes. When using the tools to develop your applications, the tools will change the update trigger from *polled* to *mbean*. This allows the tools to influence when the server discovers updated files based on when files are saved and whether or not they are actively being edited.

1.4.3 Using loose configuration for applications

In a loose configuration, an application can be distributed across arbitrary locations on disk, and yet run as if it existed in a single, flat directory structure. Tools such as Rational® Application Developer (RAD) and WebSphere Application Server Developer Tools use loose configuration to allow users to run their applications directly from the projects in their Eclipse workspace. This saves disk space, improves performance, and gives developers more immediate feedback when they make changes, because it avoids copying files between the Eclipse workspace and the server definition.

1.4.4 Configuring an application's context root

The context root is the entry point of a web application. The context root for an application is determined using the following criteria in order of precedence:

1. The `context-root` attribute of an application declared in `server.xml`
   `<application context-root="rootOne" location="..."/>`
2. The `context-root` element in the `application.xml` file in an EAR application
   `<context-root>rootTwo</context-root>`
3. The `context-root` element in the `ibm-web-ext.xml` in a web application
   `<context-root uri="rootThree" />`
4. The `name` of the declared web application in `server.xml`
   `<application name="rootFour" ... />`
5. The `Web-ContextPath` in the bundle manifest of an OSGi WAB
   `Web-ContextPath: /rootFive`
6. The directory or file name of an artifact in the `dropins` directory
   `${server.config.dir}/dropins/war/rootSix`
   `${server.config.dir}/dropins/rootSix.war`

1.4.5 Compatibility with WebSphere Application Server

The Liberty profile is a part of WebSphere Application Server and shares core technologies with the full profile. Applications developed on version v8.5 of the Liberty profile will run on version v8.0 or v8.5 of the full profile. There are differences between the Liberty profile and full profile. The difference is mostly focused on the definition and use of classloading or shared libraries. Section 3.3, "Controlling class visibility in applications" on page 69 provides an overview of how classloading and shared library support work in the Liberty profile. That overview gives special regard to management and use of third-party libraries.

Chapter 8, “From development to production” on page 139 discusses strategies for promoting applications developed using the Liberty profile to production systems.

2

Installation

There are several ways to install the Liberty profile. In this chapter, we focus on the installation methods most commonly used for setting up a development environment.

The Liberty profile has been designed for easy operation using the provided scripts and direct editing of the server configuration XML files. The development tools provide an integrated development environment (IDE) for application development as well as plenty of assistance and short-cuts to aid creating, modifying and controlling Liberty profile servers. The development tools make development easier but are not strictly required and if you prefer you can just install the runtime.

The chapter contains the following sections:

- Install the Developer Tools
- Install the Liberty profile
- Configuring the server runtime JDK
- Starting and stopping a Liberty profile server

For additional installation methods, including installation for production, refer to the following information center website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/twlp_inst.html

2.1 Install the Developer Tools

The IBM WebSphere Application Server V8.5 Liberty Profile Developer Tools are a lightweight set of tools for developing, assembling, and deploying applications to the WebSphere Application Server V8.5 Liberty profile. These tools should be installed into your Eclipse workbench. Additionally, you can use the IBM Rational Application Developer V8.5, which has the IBM WebSphere Application Server V8.5 Liberty Profile Developer Tools integrated.

For supported Eclipse versions and additional installation methods, refer to the following information center website:

`http://publib.boulder.ibm.com/infocenter/radhelp/v8r5/topic/com.ibm.rad.install.doc/topics/t_install_wdt.html`

2.1.1 Installation from Eclipse Marketplace

You can install the Liberty profile developer tools directly from the Eclipse Marketplace using the following steps:

1. Start the Eclipse integrated development environment (IDE) for Java EE Developers workbench.
2. Click **Help** → **Eclipse Marketplace.**
3. In the Find field, type `WebSphere` and then click **Go**. This will generate a list of results, as shown in Figure 2-1 on page 21.

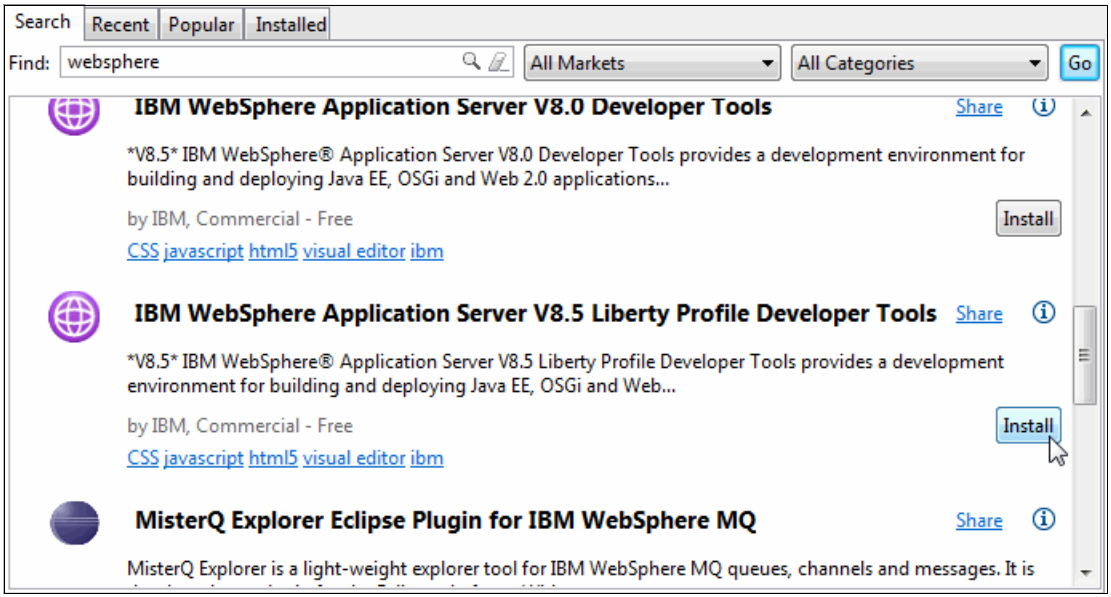


Figure 2-1 Eclipse Marketplace - selection of Liberty tools

4. In the list of results, locate IBM WebSphere Application Server V8.5 Liberty Profile Developer Tools and then click **Install**.
5. On the Confirm Selected Features page, expand the parent nodes and select the features that you want to install, as shown in Figure 2-2. When you are finished, click **Next**.

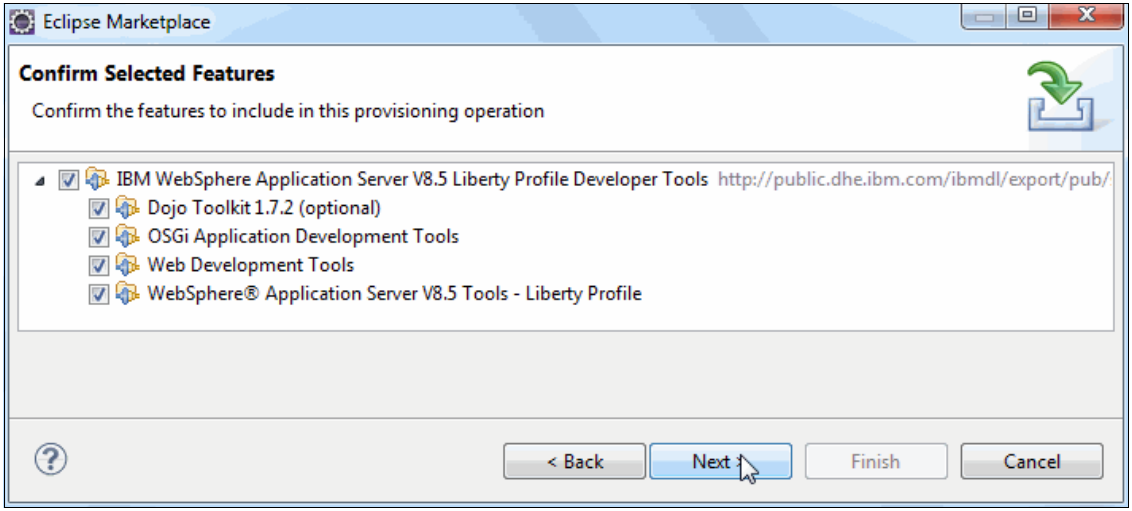


Figure 2-2 Eclipse Marketplace feature selection

6. On the Review Licenses page, review the license and if you agree to the terms, click **I accept the terms of the license agreement.** Then click **Finish,** as shown in Figure 2-3. The installation process will then start.

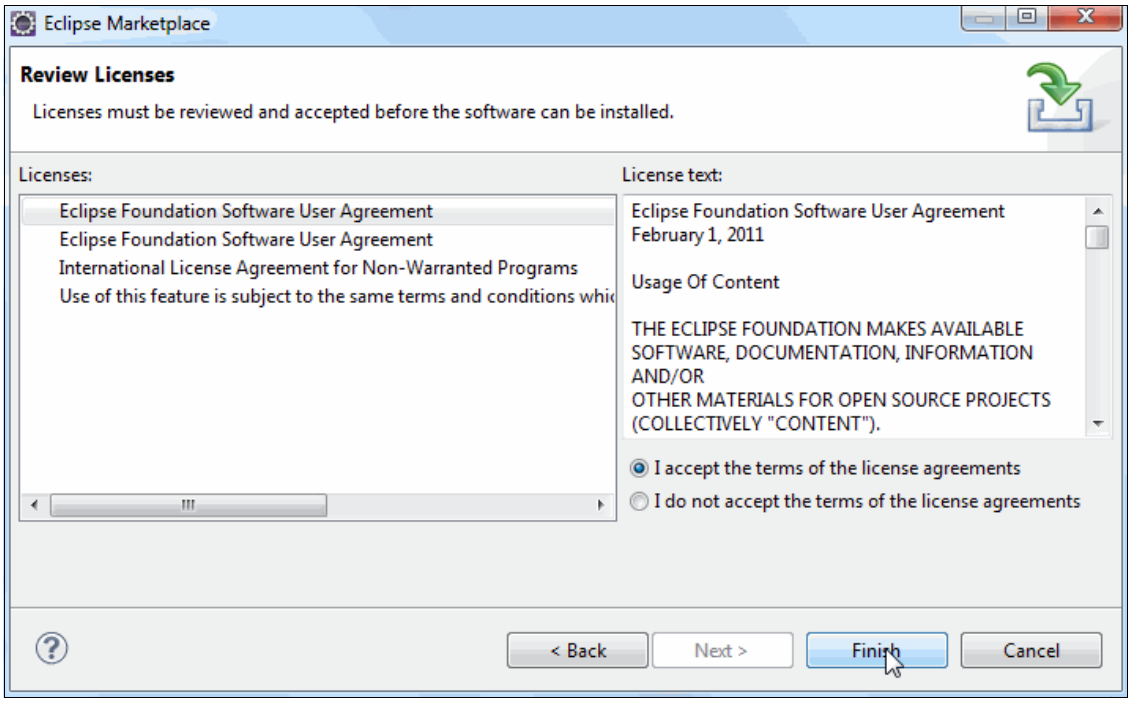


Figure 2-3 Eclipse Marketplace license agreement

> **Tip:** During the installation, a security warning dialog box might appear stating:
>
> "Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the install?"
>
> You can safely ignore the message and click **OK** to continue.

7. When the installation process completes, restart the workbench.

2.1.2 Installation from the WASdev community site

You can install the developer tools using the links provided on the WASdev community site, as long as you are using Eclipse 3.7.2 or later. Use the following steps to download the toolset:

1. Start the Eclipse IDE for Java EE Developers workbench.
2. Open your web browser to `http://wasdev.net` and click the **download** tab.
3. Select the **WebSphere Application Server Developer Tools** from the downloads page, as shown in Figure 2-4.

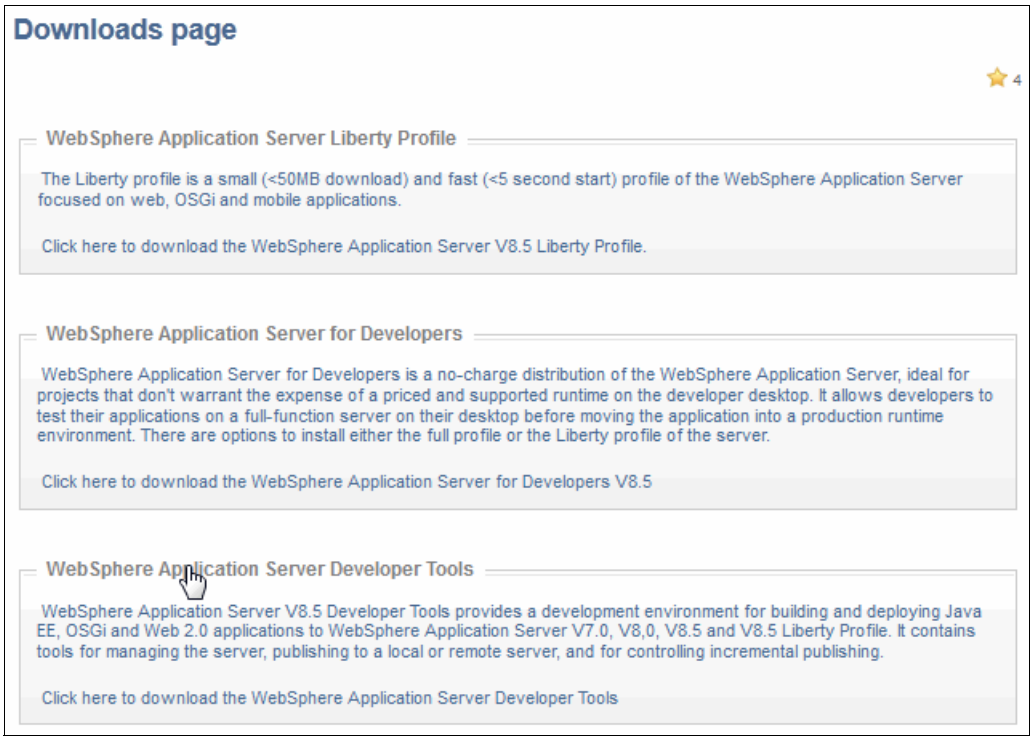


Figure 2-4 Download selection page on WASdev.net

4. Locate the Install icon for WebSphere Application Server V8.5 Liberty Profile, as shown in Figure 2-5 on page 24.

Figure 2-5 WASdev install icon

5. Select and drag the **Install icon** to your Eclipse workbench and drop it onto your toolbar.
6. On the Confirm Selected Features page, expand the parent node and select the features that you want to install, as shown in Figure 2-6. When you are finished, click **Next**.

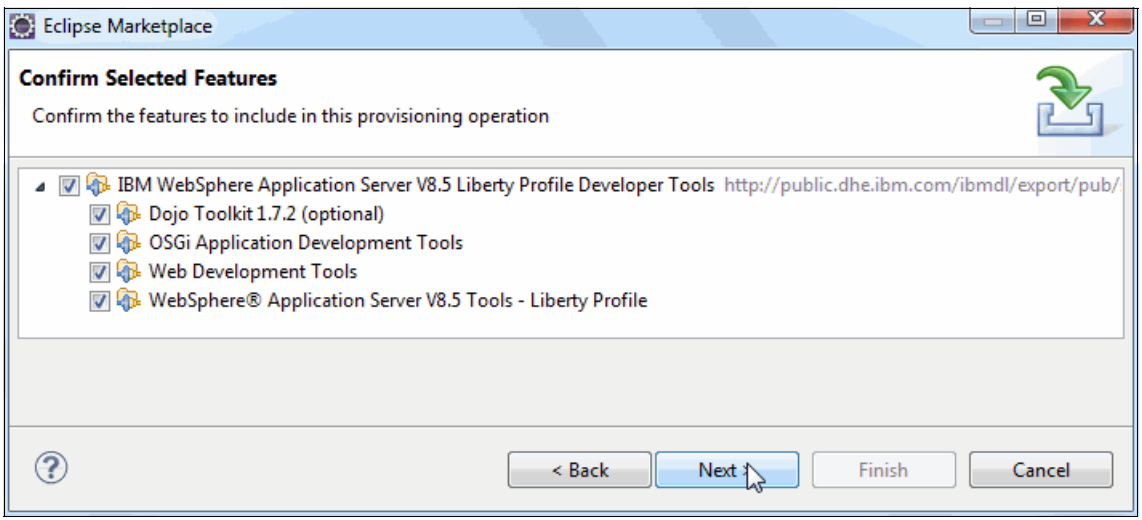


Figure 2-6 Eclipse Marketplace feature selection

7. On the Review Licenses page, review the license and if you agree to the terms, click **I accept the terms of the license agreement.** Then click **Finish**, as shown in Figure 2-7 on page 25. The installation process will then start.

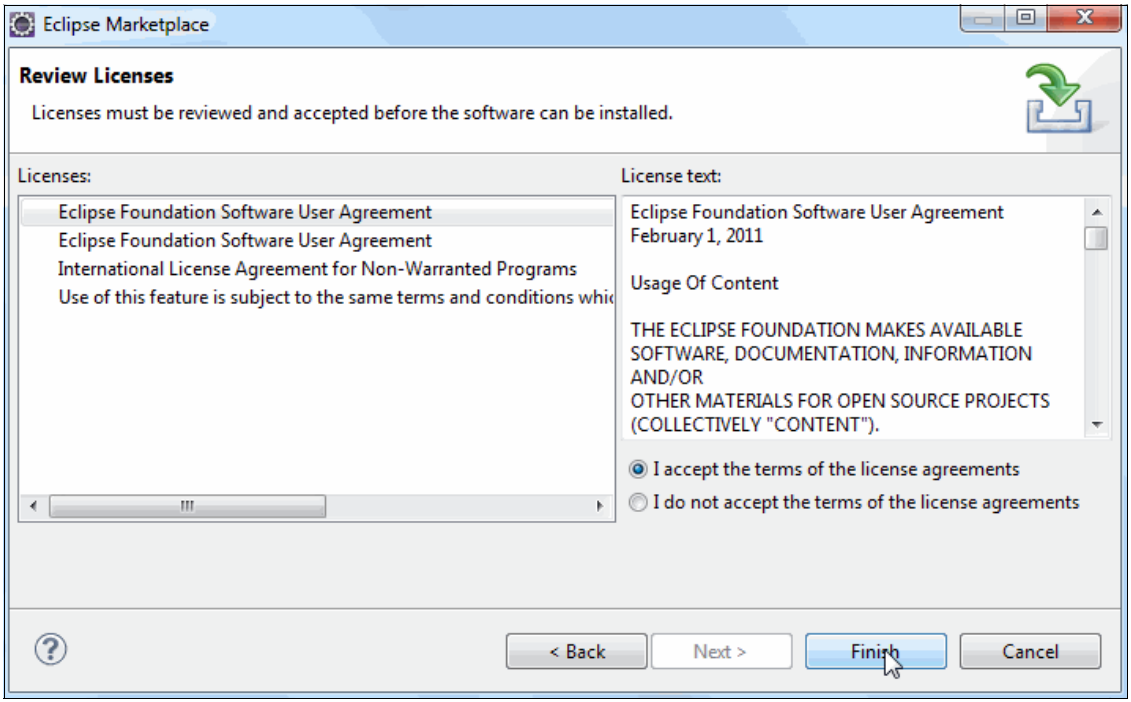


Figure 2-7 Eclipse Marketplace license agreement

> **Tip:** During the installation, a security warning dialog box might appear stating the following:
>
> "Warning: You are installing software that contains unsigned content. The authenticity or validity of this software cannot be established. Do you want to continue with the install?"
>
> You can safely ignore the message and click **OK** to continue.

8. When the installation process completes, restart the workbench.

2.1.3 Installation from downloaded installation files

You can install the IBM WebSphere Application Server V8.5 Liberty Profile Developer Tools into an existing Eclipse workbench. This is facilitated by using installation files you download to your computer.

If you run the system disconnected from the Internet, note that Eclipse will require some prerequisite files. For detailed instructions on how to perform this install and details about the additional prerequisite Eclipse files, refer to the following information center website:

http://publib.boulder.ibm.com/infocenter/radhelp/v8r5/topic/com.ibm.rad.install.doc/topics/t_install_wdt.html

2.2 Install the Liberty profile

In order to serve your applications, you need to install the Liberty profile runtime environment. The environment is the same whether you are using it for development, test, or production. The WebSphere Developer Tools integrate with the Liberty profile to give you a seamless, integrated development environment that can aid you in developing and testing your applications.

2.2.1 Installation using the Liberty profile developer tools

The Liberty profile runtime environment can be installed directly from within the WebSphere Liberty Profile Developer Tools. Installing in this manner will also create an initial server.

Tip: During the configuration of the runtime and server you will be prompted to provide names for similar items which can cause confusion. The following are the names you will be asked to define, in the order you will encounter them:

Server Host Name	The name of the machine running the server or local host.
Server Name	The name of the server as it will be known by Eclipse. This should not be confused with the actual server created in the Liberty profile.
Liberty Profile Server Name	The name of the server that will be created in the Liberty profile runtime environment. This is your actual Liberty Profile Server.

The following steps for installing the Liberty profile and server assume that you have already installed the tools described in 2.1, “Install the Developer Tools” on page 20:

1. Start Eclipse with WebSphere Developer Tools installed (or Rational Application Developer V8.5).
2. Select **File** → **New** → **Other**.

3. In the Wizards text box, type `server`. Select the **server icon** and then click **Next** as illustrated in Figure 2-8.

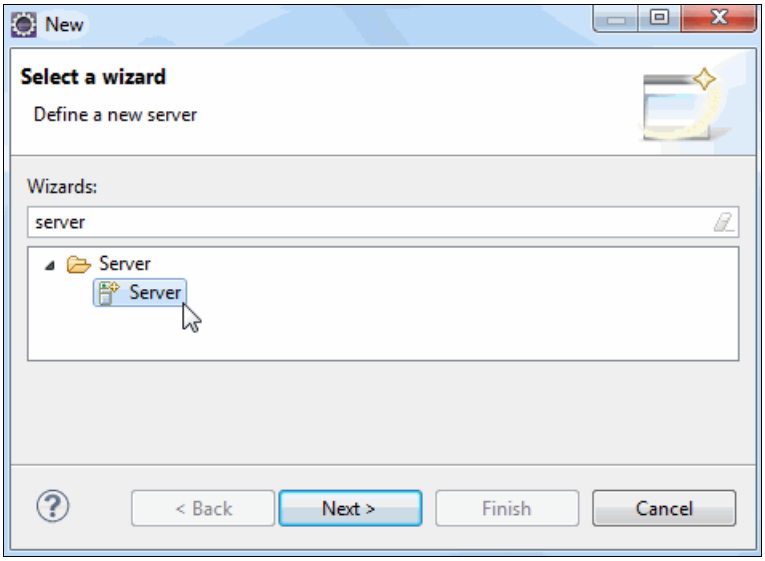


Figure 2-8 Selecting the server install wizard

4. Select **WebSphere Application Server V8.5 Liberty Profile** from the available server list, as shown in Figure 2-9 on page 28. Provide the *Server's host name* and the *Server name* that will be visible in the Eclipse workbench, then click **Next**.

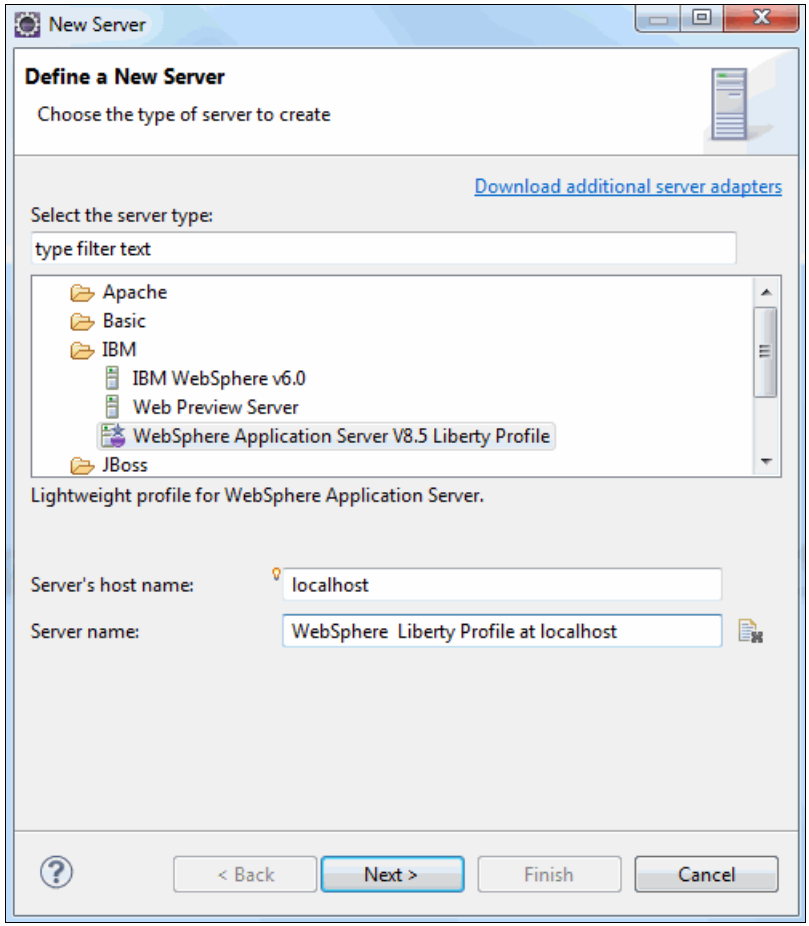


Figure 2-9 Defining the Liberty profile for a new server

5. If you have already installed a Liberty profile runtime environment, you can use the browse button to select the Liberty profile runtime install location and continue with step 9 on page 30. In our example we do not have a Liberty runtime environment installed so we need to install one. To do so, click the **Download or install** link, as shown in Figure 2-10 on page 29.

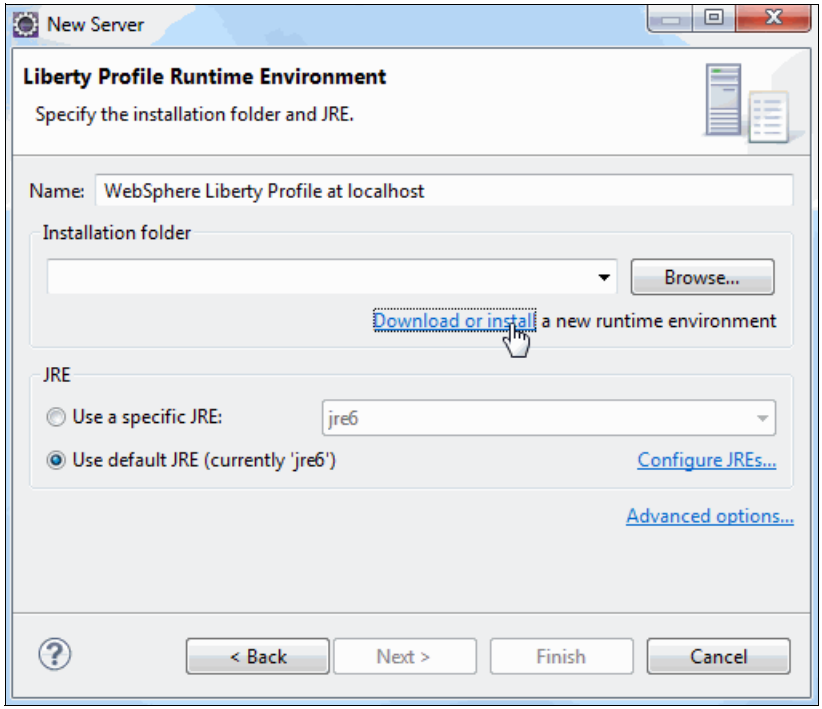


Figure 2-10 Selecting to download or install a new runtime environment

6. If you have previously downloaded a Liberty profile runtime archive file from Passport Advantage® or WASdev.net, you can select the downloaded archive using the first option. In our example, we are downloading directly from the IBM website. We chose the second option and clicked the **WebSphere Application Server V8.5 download site** and then clicked **Next** (Figure 2-11 on page 30).

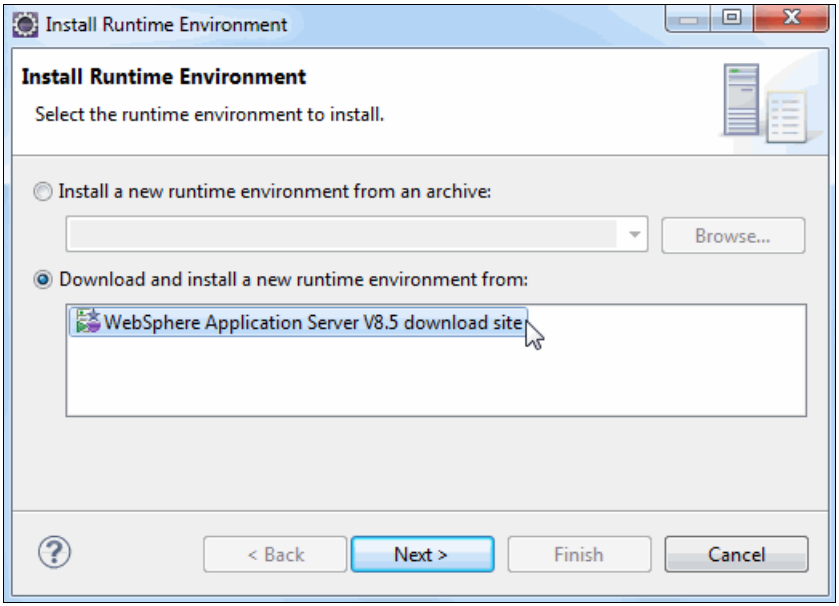


Figure 2-11 Selecting to download from the IBM site

7. Accept the License agreement, then click **Next**.
8. When asked for the installation folder, select the folder you want to install the Liberty profile runtime environment into, and then click **Finish**. The Liberty profile runtime environment will now be downloaded and installed.
9. Return to Figure 2-10 on page 29 and click **Next** to continue. Optionally, you can choose the JRE that you want the Liberty profile to use.
10. Now that you have a runtime environment you will be asked to select the Liberty profile server to use. If you have a Liberty profile server already defined, select it from the drop-down list and proceed to step 12 on page 31. If there is no Liberty profile server defined, or to create a new server, click **New** (Figure 2-12 on page 31).

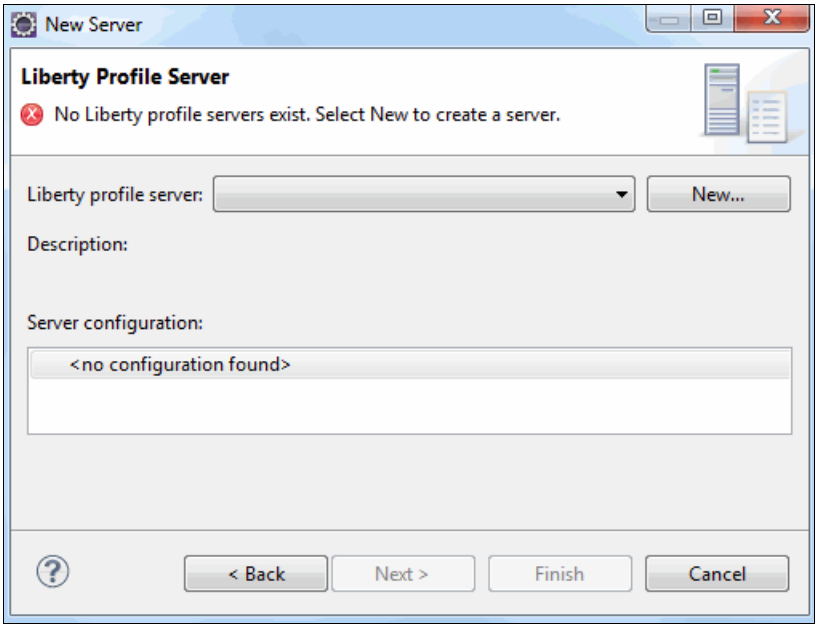


Figure 2-12 The Liberty profile server window

11. Enter the name for your new Liberty profile server and click **Finish**. A new server will be created under your runtime environment.
12. The Liberty profile server page will now show the current configuration of the server. The newly defined server shows jsp-2.2 as enabled and the default http and https ports (Figure 2-13 on page 32). To finalize the setup click **Finish**.

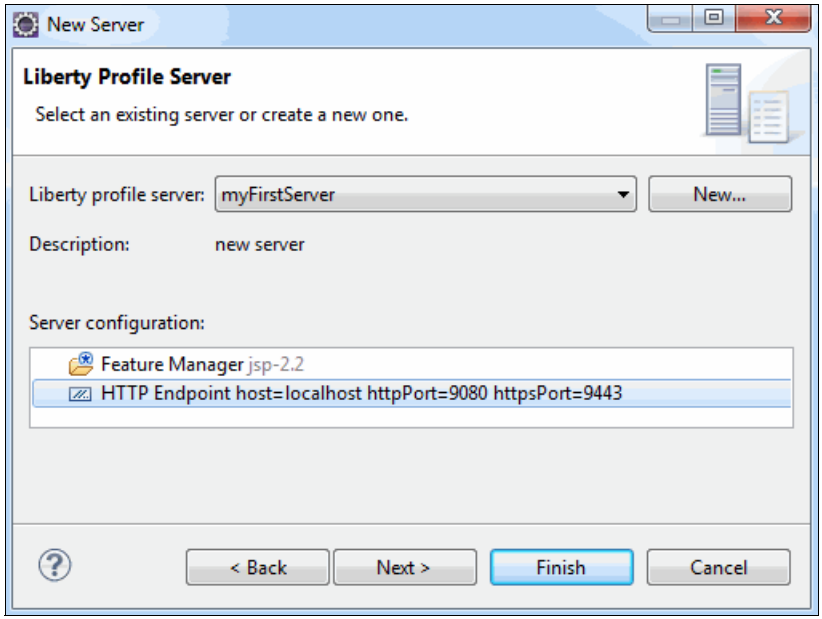


Figure 2-13 Details of the new server configuration

13. When setup is complete, you will see the new server and runtime environment displayed in the Servers view of your Eclipse workbench. Figure 2-14 shows the newly created server expanded.

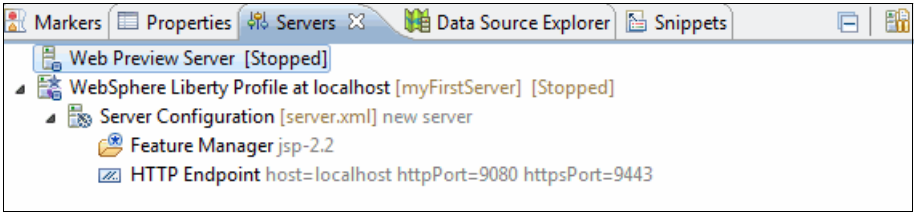


Figure 2-14 The new server shown in the Servers view

2.2.2 Installation using the command line

If you have obtained the Liberty profile runtime environment from WASdev.net or through Passport Advantage you will have downloaded a jar file. The jar file has a name similar to `wlp-developers-8.5.0.0.jar`. Always download the latest version.

Note: If you have obtained an edition other than the developers edition, the file name will contain your edition name rather than *developers*.

Installing the runtime environment

To install the Liberty profile you need to extract the archive file using the following steps:

1. Run the following command to extract the contents of the Liberty archive:

   ```
   java -jar wlp-developers-8.5.0.0.jar
   ```

2. Press x to skip reading the license terms or press Enter to view them.
3. Press Enter to view the license agreement.
4. Press 1 if you agree to the license terms and are ready to proceed.
5. Provide the installation path for the Liberty profile, for example:

   ```
   C:\IBM\WebSphere
   ```

6. Press Enter.

The server will be installed into a `wlp` directory within the `install` directory specified in step 5 on page 33. If you did not specify an install location, it will install to the current directory.

Creating a server

The Liberty profile runtime environment does not come with a server defined. In order to create a server, you need to run the following command from the Liberty profile bin directory (`wlp/bin`):

```
server create <server name>
```

This will create a server, with the provided name, in the `usr/servers` directory.

Tip: A Liberty profile runtime environment can contain multiple servers. You do not need to install multiple runtimes onto a machine to run multiple servers.

Server name is an optional parameter. If you do not specify a server name, deaultServer is used.

Adding the new server and runtime environment into the tools

In order to use the Liberty profile developer tools with your new runtime and server, you need to add them into the tools. This can be done by following the instructions in 2.2.1, “Installation using the Liberty profile developer tools” on page 26. Then select your runtime and server at the relevant steps.

2.2.3 Installation on z/OS

IBM Installation Manager is used to install the Liberty profile on z/OS using a part of the com.ibm.websphere.zOS.v85 package offering. To install the Liberty profile, install this package with the Liberty feature, using the following **imcl** command:

```
/usr/lpp/InstallationManager/V1R5/tools $ imcl install
com.ibm.websphere.zOS.v85,liberty -installationDirectory
/usr/lpp/zWebSphere/V8R5 -repositories
/usr/lpp/InstallationManagerRepository/HBBO850 -acceptLicense
```

The following folders will be created during the installation process:

- bin
- java64
- lib
- wlp
- java
- lafiles
- properties

If only core.feature is selected, which is the default, the Liberty profile will not be installed. If core.feature is selected along with the Liberty profile feature, then both WebSphere Application Server and Liberty profile will be installed.

After the WebSphere Application Server is installed without Liberty, it is not possible to add the Liberty profile feature. You will get an exception, shown in Example 2-1.

Example 2-1 Error adding Liberty profile to existing WebSphere Application Server installation on z/OS

```
CRIMA1155E ERROR: Error modifying.
ERROR:   The WebSphere Application Server and WebSphere Application
Server Liberty Profile features cannot be added to or removed from an
existing package group after installation.
```

Installation Manager cannot modify the application server features of an existing package group. To add features, use Install and include them in a new package group. To remove features, use Uninstall to completely remove the entire package group.

Important: On z/OS, create your Liberty server configurations in a location independent of the installed Liberty profile. Do not use the usr subdirectory (wlp/usr).

2.2.4 The Liberty profile runtime environment and server directory structure

The location of the various files and directories that make up the Liberty profile runtime environment and server are important. You will need to become familiar with these files to configure and deploy applications to the server.

Runtime directory structure

Figure 2-15 provides an overview of the directory structure used by the Liberty profile runtime environment. The directory structure and location of files are essential for the healthy running of your servers.

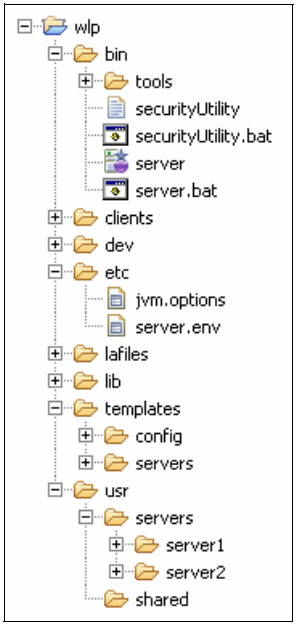


Figure 2-15 The Liberty runtime directory structure

The most important Liberty runtime directories are:

- `bin`

  This directory contains scripts used to manage the Liberty profile server and server instances

- `etc`

  This directory is optional. It can be used for customization of the whole Liberty profile installation. The configurations defined by files in this directory will apply to all Liberty profile servers.

- `lib`

  This directory contains the Liberty runtime libraries.

- `templates`

  This directory contains sample configuration files and a sample `server.xml` file for the Liberty profile.

- `usr`

  This directory contains the server instances with their configuration and applications and any resources that can be shared between servers. The `usr` directory is user-owned and as such will not be modified by any service releases.

Server directory structure

Similar to the runtime directory structure, it is essential that you become familiar with the location and layout of the Liberty profile server files and directories, as shown in Figure 2-16.

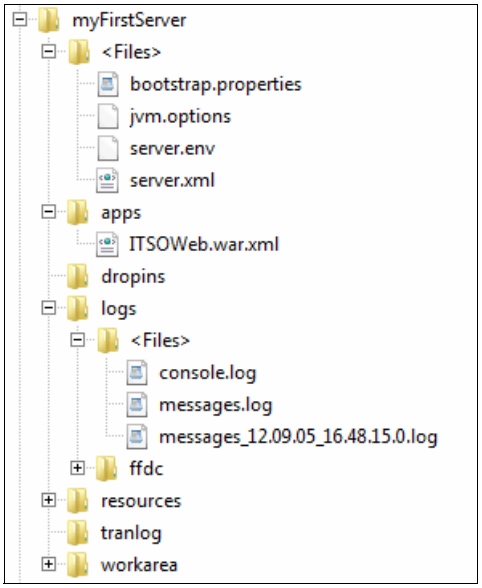


Figure 2-16 The Liberty server directory structure

The important Liberty profile server directories are:

- `apps`

  This directory is optional. It can contain deployed applications or application descriptors if they are deployed using the WebSphere Developer Tools.

- `dropins`

  Applications can be added and removed to the liberty server simply by dropping them into this folder. This directory is constantly monitored by the server to identify any changes and add or remove an application when the changes occur.

- `logs`

  Contains the logs produced by the Liberty profile server. By default, this is the place where the trace or message logs will be written. It will also contain the First Failure Data Captures (FFDC).

- `resources`

  Contains additional resources for the Liberty profile server instance. For example, keystores generated by the Liberty profile server will be located in this directory.

- `tranlog`

  Contains the transactional logs that are produced by the server runtime and the applications. The transactional logs are used to commit or rollback transactional resources in the event of server failure.

- `workarea`

  Contains the Liberty profile server operational files; is created during the first server run.

2.3 Configuring the server runtime JDK

The Liberty profile requires a Java Runtime Environment (JRE) to run in. The Liberty profile server will run on Java version 6 or above provided by any vendor. A full list of supported Java versions is noted in the following System Requirements document:

http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg27028175

The following sections describe the ways you can define which JRE is used.

2.3.1 Defining the JRE from within Eclipse Workbench

When running a server through the Eclipse Workbench, the JRE to be used is defined in the server properties. When you created the server in Eclipse, you had the option to define the JRE. To change the JRE, follow these steps:

1. Switch to the Servers view in the Eclipse Workbench.

2. Right-click your server and select **Open,** as shown in Figure 2-17.

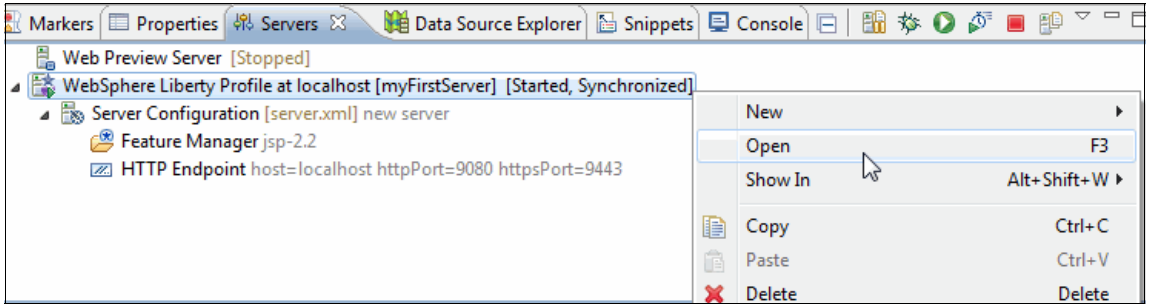


Figure 2-17 Opening a Server Properties panel

3. Select the **Runtime Environments** link as shown in Figure 2-18.

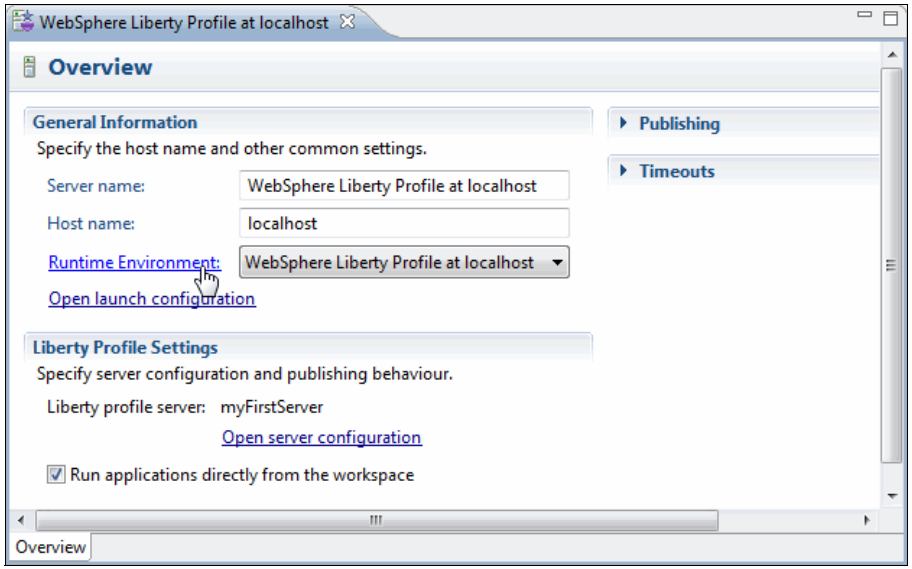


Figure 2-18 Opening the Runtime Environment properties

4. A window will be displayed allowing you to make changes to the runtime environment, including changing your JRE. After you are satisfied with your changes, click **Finish**.
5. Save the changes to the Server properties. The new settings will be used when you next start the server.

Caution: If you have created a `server.env` file that defines the JRE (either for the server or for the runtime) this will take preference over the Eclipse setting. You will need to remove this file in order for the Eclipse setting to work.

2.3.2 Configuring the system JRE

When starting the Liberty server using the command line, the server needs to know where to find its JRE. In the absence of any server or runtime configuration files that define the JRE, the system JRE will be used.

On Windows systems you can use Example 2-2 to set the JAVA_HOME property, so the Liberty server uses your own JRE installation directory.

Example 2-2 Setting the JRE for Liberty on Windows

```
set JAVA_HOME=C:\IBM\jre6
set PATH=%JAVA_HOME%\bin;%PATH%
```

On Linux Systems you can use Example 2-3 to set the JAVA_HOME property.

Example 2-3 Setting the JRE for Liberty on Linux

```
export JAVA_HOME=/opt/IBM/jre6
export PATH=$JAVA_HOME:$PATH
```

The Liberty profile runtime environment will search for the Java command in the following order of properties: JAVA_HOME, JRE_HOME and PATH.

2.3.3 Using server.env to define the JRE

The `server.env` file is a Liberty profile-specific configuration file that can be used to define the JRE for the servers to use. When you install the Liberty profile this configuration file does not exist, so you will need to create it. Create the file in one of the following locations:

- `${wlp.install.dir}/etc/server.env`
  The settings here will apply to all servers in the runtime environment.
- `${server.config.dir}/server.env`
  These settings are specific to the server containing the file. This `server.env` is used in preference to the runtime level when both exist.

To define the JRE, add the following line into the `server.env` file:

`JAVA_HOME=<path to your JRE>`

You can use the `server.env` file to provide any other additional environment variables that your server might need.

Caution: The `server.env` files only support key=value pairs. Variable expansion is not supported.

2.3.4 Specifying JVM properties

You can use `jvm.options` files at the runtime and server levels to specify jvm-specific start options, for example, -X arguments. The options are applied when you start, run, or debug the server. Be sure to specify only one option per line.

When you install the Liberty profile, this configuration file does not exist, so you will need to create it. Create the file in one of the following locations:

- `${wlp.install.dir}/etc/jvm.options`
  The settings here will apply to all servers in the runtime.
- `${server.config.dir}/jvm.options`
  These settings are specific to the server containing the file. This `jvm.options` file is used in preference to one at the runtime level when both exist.

2.4 Starting and stopping a Liberty profile server

You can control the state of the Liberty profile server either from the command line or through the Eclipse Workbench. We describe the two methods in this section.

2.4.1 Starting and stopping the server using the command line

The server can be controlled from a single **`server`** command located in the `${wlp.install.dir}/bin` directory. The following commands can be used to start and stop the server:

- `server start <servername>`
  Starts the server running in the background
- `server run <servername>`
  Starts the server running in the foreground and writes the console output to the window. To stop the Liberty profile server in this mode, press Ctrl+c or kill its process or run server stop from another command window.

- `server debug <servername>`
  Starts the server in debug mode.
- `server stop <servername>`
  Stops the server.
- `server status <servername>`
  Displays the current state of the server.

Hint: If Eclipse Workbench is open and it has the Liberty profile server already defined, it will automatically update. Updates reflect the current status of the server and show the output from the server in the console window.

2.4.2 Starting and stopping the server from the Eclipse Workbench

You can control the state of the Liberty profile server from directly within the Eclipse Workbench. This can be done from the Server view. Either right-click the server and select the required option, or click the icons attached to the Server view, as shown in Figure 2-19.

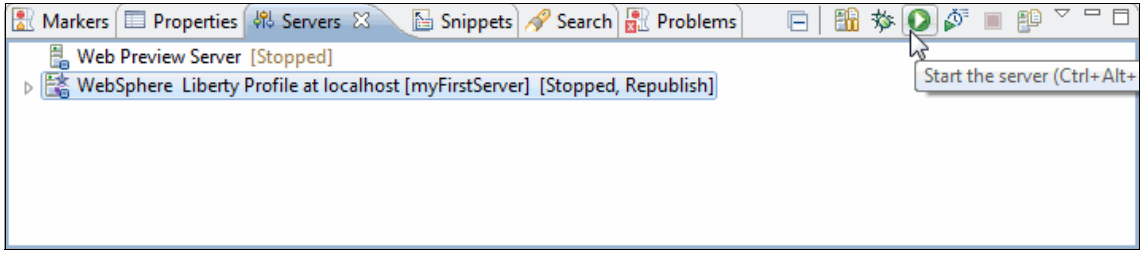


Figure 2-19 Starting the server using the start button

The following buttons are available to control the server:

 Starts the server in debug mode.

 Starts the server.

 Stops the server.

3

Developing and deploying web applications

IBM WebSphere Application Server V8.5 Liberty Profile provides a simplified environment for developing and deploying applications. Whether working in the Liberty profile developer tools or in a third-party text editor, you can quickly build, run, and update applications. Development overhead, such as configuration and server restarts, are kept to a minimum to reduce the amount of time it takes to develop new applications. In this chapter we describe the benefits of developing applications in the Liberty profile environment.

The chapter contains the following sections:

- Developing applications using the Liberty profile developer tools
- Developing outside the Liberty developer tools
- Controlling class visibility in applications

3.1 Developing applications using the Liberty profile developer tools

The Liberty profile developer tools provide a lightweight, rich environment for developing applications. Using the developer tools simplifies the creation of applications by providing wizards and graphical design tools for JavaEE applications. Server configuration tasks are simplified using the tools. In many cases, the tools can infer what configuration is needed based on the application and update it automatically. Working in the Liberty profile developer tools environment also enables iterative development by republishing applications when you make changes in the integrated development environment (IDE).

In this section, we describe an iterative development scenario involving several technologies supported by the Liberty profile server.

3.1.1 Using the tools to create a simple servlet application

In this section we describe how to develop and deploy a simple servlet application using either the Liberty profile developer tools or a simple text editor. Following the procedure, described in this book, shows the simplicity of developing and updating applications in the Liberty environment.

Liberty profile supports the JavaEE Servlet 3.0 API. This book assumes some knowledge of JavaEE technologies, so the actual coding of the example servlet is not discussed here.

The Liberty profile developer tools provide a rich development environment for building web applications. In this section we take you through the end-to-end process of developing, deploying, and updating an application.

Getting started

Before you begin, make sure you have installed the Liberty profile server and tools as described in 2.1, “Install the Developer Tools” on page 20. The following steps outline how to create a new project for later deployment:

1. Begin by creating a new web project in the Liberty developer tools using the `Create a Web Project` wizard from the eclipse toolbar.
2. In the first window, give the project the name `ITSOweb` and select the **Simple** project template.
3. Leave Programming Model as the default value, `Java EE`. Clicking **Next** takes you to the Deployment options.

4. Under Target Runtime, make sure that **WebSphere Application Server V8.5 Liberty Profile** is selected. Leave all other options as the default values, and click **Finish**.

Build a servlet application

Now add a servlet to the ITSOweb web project by completing the following steps:

1. Begin by creating a new servlet using the Create Servlet wizard. Use `com.itso.example` for the package name and `HelloITSO` for the servlet class name. All other values can be left as the defaults. The panel in Figure 3-1 shows the Create Servlet wizard.

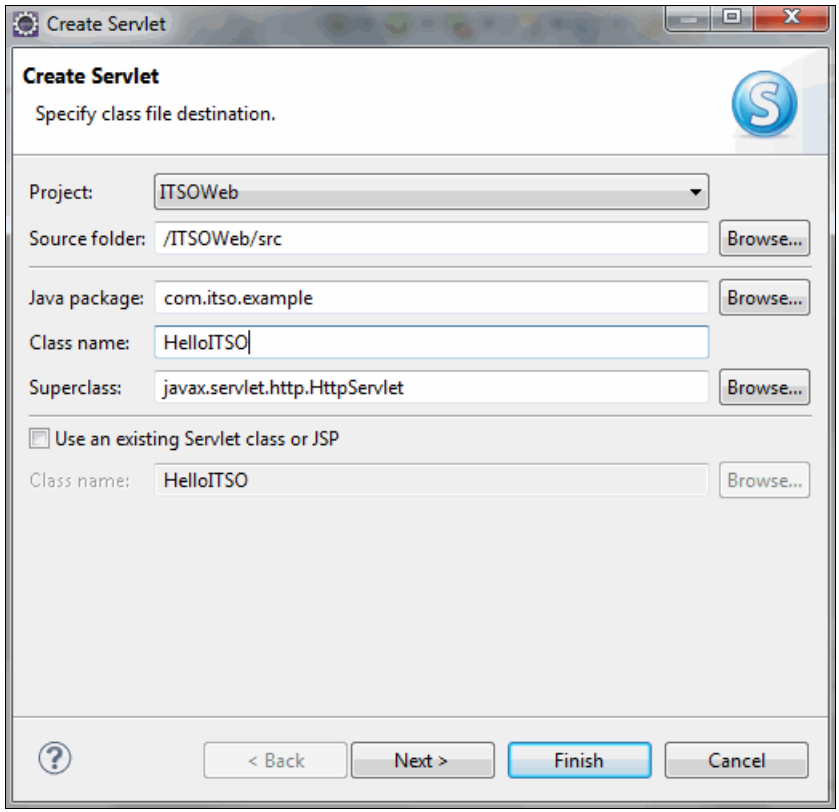


Figure 3-1 Create Servlet wizard

2. After finishing the wizard, the file `HelloITSO.java` will be included in the ITSOWeb project. Open the file and add some simple output to the servlet's **`doGet`** method. Figure 3-2 on page 46 shows an example **`doGet`** method.

```
protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    PrintWriter pw = response.getWriter();
    pw.println("<BODY>");
    pw.println("<H2>Hello, Liberty developer</H2>");
    pw.println("</BODY>");
}
```

Figure 3-2 Example doGet method for the HelloITSO servlet

For more information about developing servlets and other dynamic web content, see the Liberty profile developer tools at the following information center website:

http://publib.boulder.ibm.com/infocenter/radhelp/v8r5/topic/org.eclipse.wst.webtools.doc.user/topics/twsrvwiz.html

Deploy the application

Prior to deploying the application, ensure that you have created a Liberty Profile Server in the development environment as described in 2.2.1, “Installation using the Liberty profile developer tools” on page 26. To deploy the application complete the following steps:

1. In the Servers view, locate your Liberty Profile Server. Right-click the server and select **Add and Remove**.
2. In the resulting panel, select the ITSOWeb project and click **Add**, then click **Finish**. The web project will now be available on the server. You should see output similar to Figure 3-3 in the console. Note that the application was automatically assigned the context root matching the project name of ITSOWeb. Also note that the application start is extremely fast.

```
[AUDIT   ] CWWKG0016I: Starting server configuration update.
[AUDIT   ] CWWKT0016I: Web application available (default_host): http://localhost:9080/ITSOWeb/*
[AUDIT   ] CWWKZ0001I: Application ITSOWeb started in 0.8 seconds.
[AUDIT   ] CWWKG0017I: The server configuration was successfully updated in 0.09 seconds.
```

Figure 3-3 Output from deploying an application

Result: Note the extreme speed of the application start and that it was automatically assigned the context root matching the project name of ITSOWeb.

Verify the application

To verify the application, open a web browser and go to:

http://*localhost*:9080/ITSOWeb/HelloITSO

You should see the following output syntax:

```
Hello, Liberty developer
```

Update the application

The Liberty profile server supports hot deployment for all applications. To verify, change the servlet to output a different message in its **doGet** method. When you save the file, the changes will automatically be published to the server. In the console output, notice that the application was stopped, updated, and made available again almost instantaneously. Verify that your changes are active by visiting the HelloITSO page again.

3.1.2 Developing and deploying a JSP application

The Liberty profile server supports the JavaServer Pages (JSP) 2.2 specification. JSP allows you to separate presentation logic from your business logic.

The procedure for creating and deploying a JSP in the Liberty profile developer tools is similar to the procedure for a servlet. Use the following steps:

1. Begin by creating a new JSP file by right-clicking the **ITSOWeb** project and selecting **New** → **JSP File**.
2. Enter `ITSOWelcome.jsp` for the name of the page and click **Next**.
3. On the Select JSP template panel, make sure that the template **New JSP File (HTML)** is selected and click **Finish**.

The ITSOWelcome.jsp page will automatically open in the Rich Page editor. By default, the editor will display a split view that shows both the JSP source and a preview of the output. Insert some text into the source editor between the body tags. Notice that the preview pane is automatically updated. Figure 3-4 on page 48 shows the JSP editor after adding a form to submit a simple input value to the HelloITSO servlet.

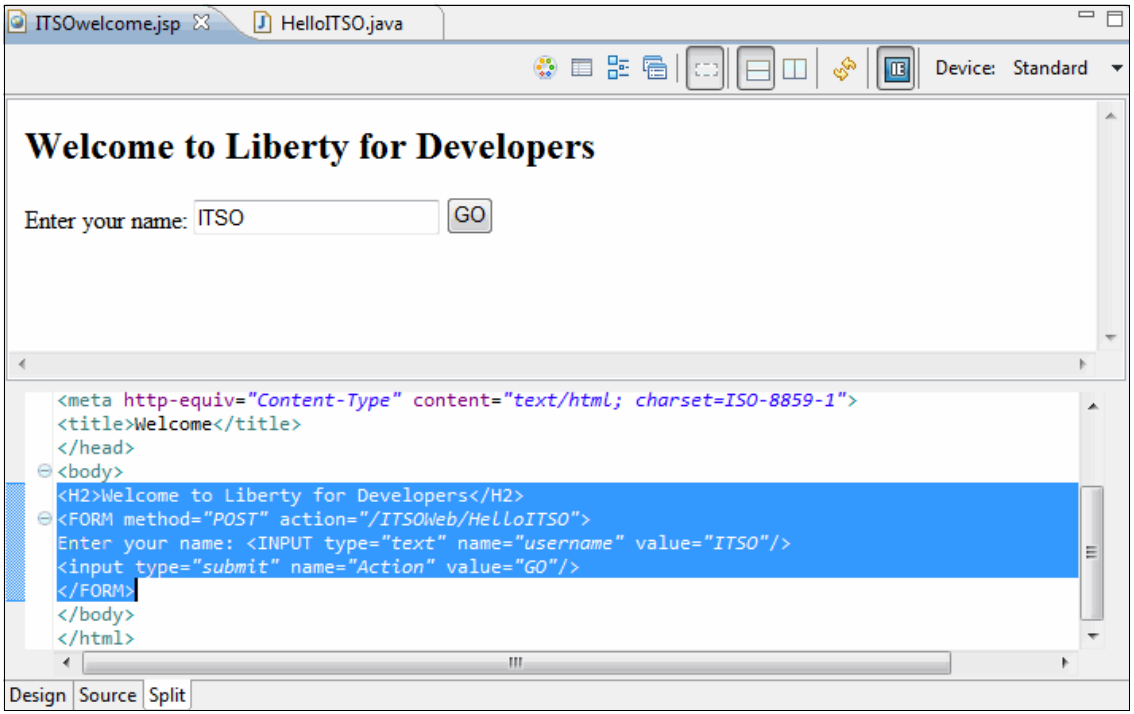


Figure 3-4 Editing the ITSOWelcome JSP file

The Rich Page editor contains several powerful features that are beyond the scope of this book. For more information, refer to the following information center website:

http://publib.boulder.ibm.com/infocenter/radhelp/v8r5/topic/com.ibm.etools.rpe.doc/topics/crpe.html

Because the ITSOWeb project is already deployed to the ITSO Example Server, there are no extra steps needed to deploy and test the new JSP file. However, to test the JSP file, shown in Figure 3-4, open the HelloITSO servlet again and add an implementation for the **`doGet`** method. For example, the code, shown in Figure 3-5 on page 49, will output a message using the value of the JSP's input field. When you are finished editing the servlet, save and close the file. Notice that the application will automatically be updated on the server.

```
protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    hello(response.getWriter(), request.getParameter("username"));
}

private void hello(PrintWriter writer, String name) {
    writer.println("<BODY>");
    writer.println("<H2>Hello, " + name + "</H2>");
    writer.println("</BODY>");

}
```

Figure 3-5 Example doPost implementation for the HelloITSO servlet

To test the application, visit:

`http://localhost:9080/ITSOWeb/ITSOWelcome.jsp`

You should be able to submit a value in the JSP file and see the output message from the servlet that reflects the value given in the input field of the JSP.

3.1.3 Developing and deploying a JSF application

JavaServer Faces (JSF) simplifies the development of user interfaces for web applications by providing the following features:

- Templates to define layout
- Composite components that turn a page into a JSF UI component
- Custom logic tags
- Expression functions and validation
- Component libraries
- XHTML page development

For more information about JSF and the features it provides, refer to the following website:

`http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/cweb_javaserver_faces.html`

JSF tools

The Liberty profile developer tools provide a rich development environment for building high quality JSF applications. In this section, we describe a simple JSF application using JSP files, a managed bean, and navigation rules. For more information about the powerful features provided by the JSF tools, refer to the following information center website:

`http://publib.boulder.ibm.com/infocenter/radhelp/v8r5/topic/com.ibm.etools.jsf.doc/topics/tjsfover.html`

Enabling JSF in a web project

To enable JSF in a web project, right-click the project and select **Properties**. Click **Project Facets** and enable **JavaServer Faces,** as shown in Figure 3-6.

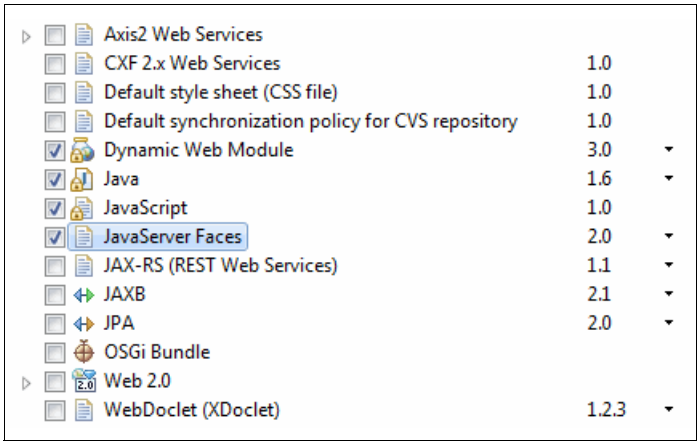


Figure 3-6 Enabling JSF in a web project

Feature enablement

To reduce overhead and resource usage, the Liberty profile server only enables features that are actively being used. When you create a server using the Liberty profile developer tools, the configuration will automatically be updated to enable servlets and JSPs. By default, the JSF application runtime is not enabled. The server configuration must be updated to run JSF applications.

When you add applications to the server or update the facets of a project deployed to the server, the Liberty profile developer tools will try to determine what features should be enabled on the server to support the application. If you update the project to add the JSF facet, the tools will prompt you to add the jsf-2.0 feature to the Liberty profile server, as shown in Figure 3-7 on page 51.

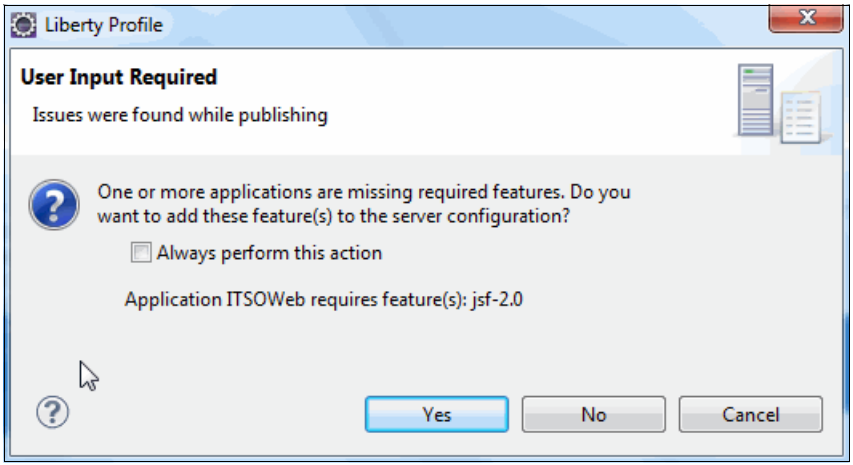


Figure 3-7 JSF feature enablement

You can verify that this has taken place by opening the `server.xml` file and confirming that `<feature>jsf-2.0</feature>` has been added. Also, the console output will show that the jsf-2.0 and beanvalidation-1.0 features were installed to the server, as shown in Figure 3-8.

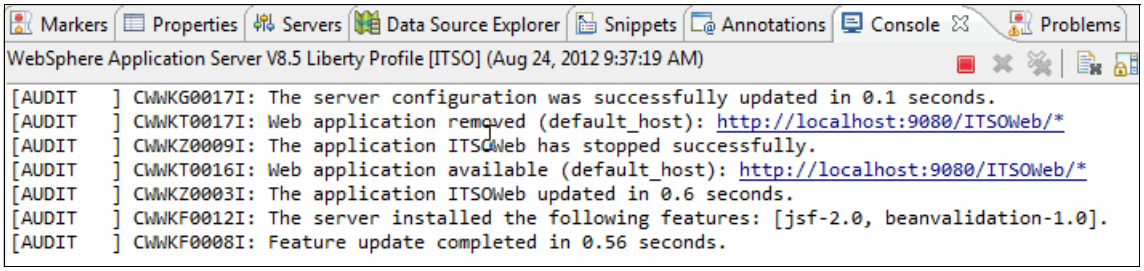


Figure 3-8 Console output after JSF feature enablement

Build a JSF application

Now that JSF is enabled in your web project, begin creating a JSF application by opening the `WebContent/WEB-INF/ faces-config.xml` file for editing in the Faces Config editor.

Add a managed bean and set properties

Complete the following steps to add a managed bean:

1. On the ManagedBean tab, click **Session** and then click **Add** to add a new session scoped managed bean.

2. In the resulting panel, select **To create a new java class for the managed bean**.
3. In the following panel, create a new class named `UserManagedBean` in the `com.itso.example` package.
4. Go through the remaining panels in the managed bean wizard by clicking **Next**, or just click **Finish**.

Figure 3-9 shows the ManagedBean configuration after the wizard has finished.

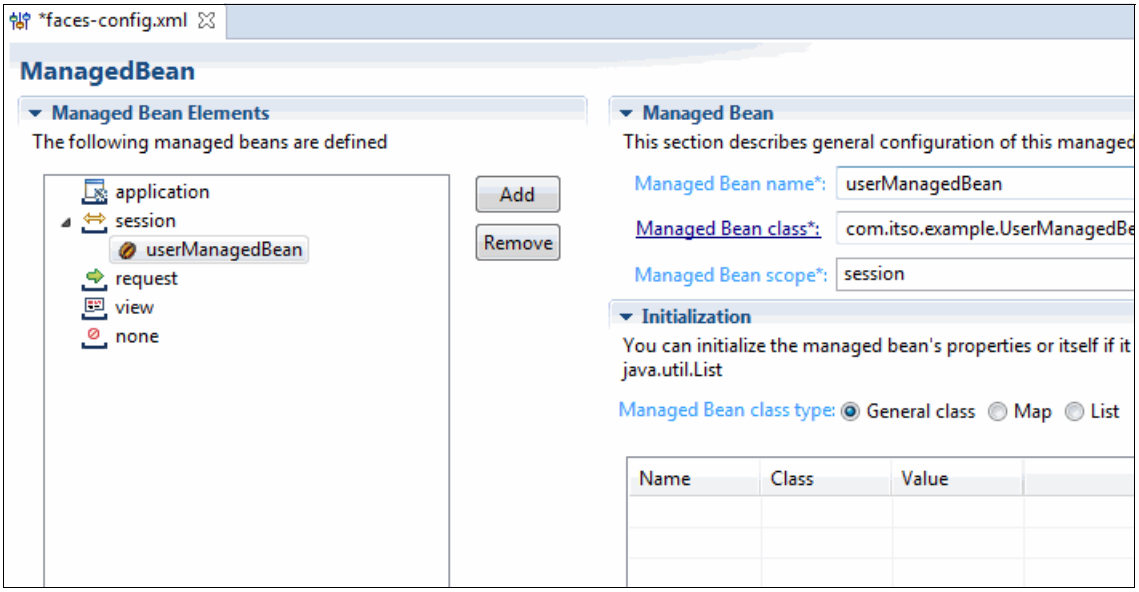


Figure 3-9 ManagedBean configuration for UserManagedBean

Now that the managed bean has been added, complete the following steps to edit the userManagedBean class:

1. Add String fields named user and location.
2. Add a getter and setter for both fields.
3. Add a method named clear that sets both fields to null and returns the string “clear”, and a method named submit that simply returns the string “submit”.

Example code for the userManagedBean class is shown in Figure 3-10 on page 53.

```
public class UserManagedBean {

    private String name;
    private String location;

    public String getLocation() {
        return location;
    }
    public void setLocation(String location) {
        this.location = location;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public String clear() {
        setName(null);
        setLocation(null);

        return "clear";
    }
    public String submit() {
        return "submit";
    }


}
```

Figure 3-10 Example UserManagedBean code

To further edit the properties of the managed bean, complete the following steps:

1. Go back to the Faces Config editor for the managed bean. You can set initial values for the managed bean's properties by clicking **Add** under the Initialization section.
2. From the property name drop-down, select the **Name property**. In the value field, enter `Liberty User` (Figure 3-11 on page 54).

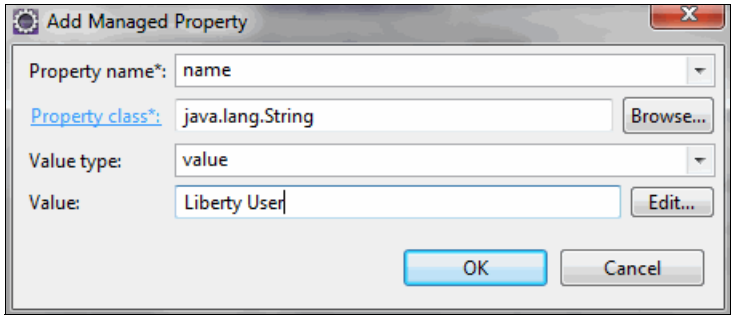


Figure 3-11 Adding a Managed Property

3. Repeat the process for the location property.

Add JSF-enabled JSP files

Create three new JSP files, `input.jsp`, `visit.jsp`, and `goodbye.jsp` using the Create JSP wizard. In the wizard's template selection page, select **New JavaServer Faces (JSF) Page (html)**.

The `input.jsp` file will ask for two input parameters, a name and a location. The values will be taken from the managed bean **`uuserManagedBean`**. Figure 3-12 shows example code to accomplish this process.

```
<f:view>
    <h2>Welcome, Liberty Developer </h2>

    <h:form>
          <BR>Enter your name:
          <h:inputText id="name" value="#{userManagedBean.name}"/>
          <BR>Enter your location:
          <h:inputText id="location" value="#{userManagedBean.location}"/>
          <BR>
          <h:commandButton action="#{userManagedBean.submit }" value="submit"/>
    </h:form>
</f:view>
```

Figure 3-12 Example code for input.jsp

The `visit.jsp` file should output the name and location value from the userManagedBean. It then displays a button to return to the previous page, and a button to quit the application. Figure 3-13 on page 55 shows example code for `visit.jsp`.

```
<f:view>
    <h:form>
        <h2>Hello, #{userManagedBean.name } from #{userManagedBean.location}</h2>
        <br>
        <h:commandButton action="#{userManagedBean.clear }" value="Go Back" />
        <br>
        <h:commandButton action="goodbye" value="Leave"/>
    </h:form>
</f:view>
```

Figure 3-13 Example code for visit.jsp

The `goodbye.jsp` file should output the name and have a command button that allows you to start over. Example code is shown in Figure 3-14.

```
<h:form>
 <h2>Goodbye, #{userManagedBean.name }</h2>
 <BR>
 <h:commandButton action="#{userManagedBean.clear}" value="Start Over"/>
 </h:form>
```

Figure 3-14 Example goodbye.jsp file

Add navigation rules and set outcome properties

To add navigation rules, complete the following steps:

1. In the Faces Config editor, click the tab labeled **Navigation Rule**. Drag each of the three JSP files you just created into the editor.
2. Create links between the files. If the palette is not already displayed, select **Window** → **Show View** → **Palette**. On the palette, click **Link**, and then click the `input.jsp` file. End the link by clicking the `visit.jsp` file. You should now see an arrow pointing from `input.jsp` to `visit.jsp`.
3. Click the **link** and view the properties in the properties view. If the properties view is not available, select **Window** → **Show View** → **Properties**. In the `from outcome` field of the properties, enter `submit`.
4. Follow the same procedure to create links between the remaining files.

To set the outcome properties, complete the following steps:

1. Create a link between the `visit.jsp` file and the `goodbye.jsp` file, and set the outcome field to `goodbye`.
2. Create a link between the `goodbye.jsp` file and the `input.jsp` file, and set the outcome property to `clear`.
3. Finally, create a link from the `visit.jsp` file back to the `input.jsp` file and set the outcome to `clear`.

Figure 3-15 shows the navigation rule panel after all the rules have been created.

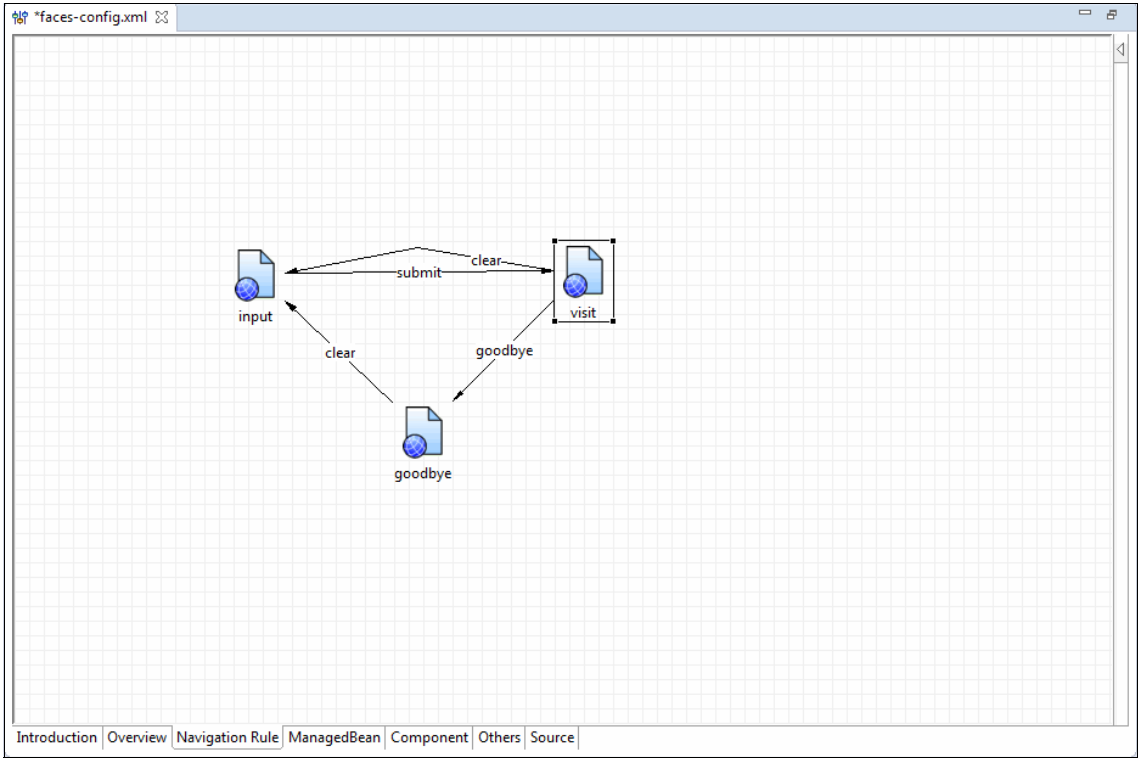


Figure 3-15 JSF navigation rules

Test the application

Access the application by visiting:

`http://localhost:9080/ITSOWeb/faces/input.jsp`

Result: Notice how the managed bean fields were initialized without having to add code in the JSP. Also notice that the application navigates to the correct pages based on the navigation rules without having to embed navigation logic in the JSP files.

3.1.4 Developing and deploying JAX-RS applications

The Java API for Restful Web Services (JAX-RS) simplifies the development of applications that use Representational State Transfer (REST) principles.

More information about JAX-RS is available at the following website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/cwbs_jaxrs_overview.html

Building a simple JAX-RS application

To build and configure a JAX-RS application in the Liberty profile developer tools, complete the following steps:

1. Enable the JAX-RS facet on the web project by clicking the **box** next to JAX-RS (REST Web Services), as shown in Figure 3-16.

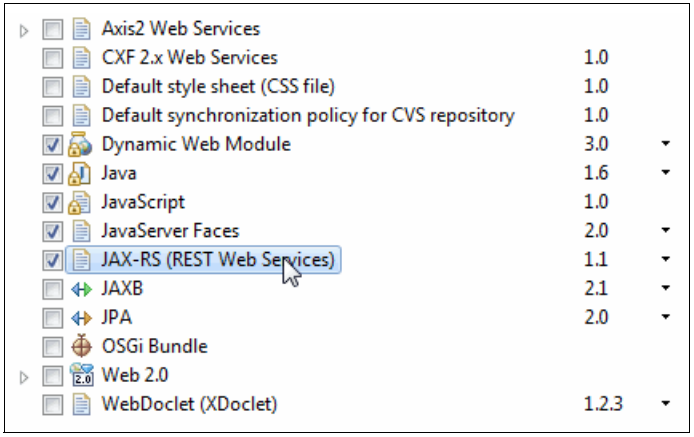


Figure 3-16 Web project facet list

2. After you click the **box** next to JAX-RS (REST Web Services) you will see a red circle below the panel with a note that further configuration is required.
3. Clicking the note will open a panel with various configuration options for JAX-RS. You can leave all of the values on this panel as their defaults and click **OK**. If you open the `web.xml` file for the web project, you will notice that a servlet and servlet mapping have been added for JAX-RS.

Feature enablement

Enabling the facet will prompt you to enable the jaxrs-1.1 feature on the server. As with JSF, the Liberty profile server will not load the JAX-RS runtime unless it is actively being used. Click **Yes** to enable the feature. If you load the `server.xml` file, in the source view, you will notice that <*feature*>jaxrs-1.1</*feature*> has now been added to the list of feature definitions.

Create an application class

A JAX-RS application class is used to define the classes used in the application. Create one now by creating a new Java class that extends `javax.ws.rs.core.Application`. The class should contain a **`getClasses()`** method that returns the classes involved in the application, as shown in Figure 3-17.

```
public class ITSOJaxrsApplication extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> classes = new HashSet<Class<?>>();
        classes.add(ITSOJaxrsExample.class);
        return classes;
    }

}
```

Figure 3-17 JAX-RS application class

Create the ITSOJaxrsExample class

To create the class for your JAX-RS, use the following steps:

1. Create a Java class named ITSOJaxrsExample or import the class that is included in the download material for this book. For more information see Appendix A, “Additional material” on page 147.
2. The class needs to be annotated with `@Path(“/example”)` to indicate that the class will be used with requests to `ITSOWeb/jaxrs/example`.
3. Add a method called **`getString`** that returns a `javax.ws.rs.Response` object. The method needs to be annotated with `@GET`. Also add a method called **`postString`** that takes a string as a parameter and returns a response object. The **`postString`** method should be annotated with `@POST`. Figure 3-18 on page 59 shows an example of the completed class.

```java
@Path("/example")
public class ITSOJaxrsExample {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public Response getString() {
        return Response.ok("Hello ITSO from JAX-RS").build();
    }

    @POST
    @Consumes(MediaType.TEXT_PLAIN)
    @Produces(MediaType.TEXT_PLAIN)
    public Response postString(String incomingMessage) {
        return Response.ok("Hello, " + incomingMessage + " from JAX-RS").build();
    }
```

Figure 3-18 JAX-RS resource class

Add a servlet initialization parameter

The servlet that was added when we updated the web project to enable the JAX-RS facet needs to know about your application class. This is done by creating an initialization parameter for the JAX-RS Servlet. To create an initialization parameter, complete the following steps:

1. Open the `web.xml` file in the design editor and right-click **JAX-RS Servlet**.
2. Select **Add** → **Initialization Parameter**.
3. On the resulting details panel, enter `javax.ws.rs.Application` in the name field and the name of your Application class in the value field, as shown in Figure 3-19.

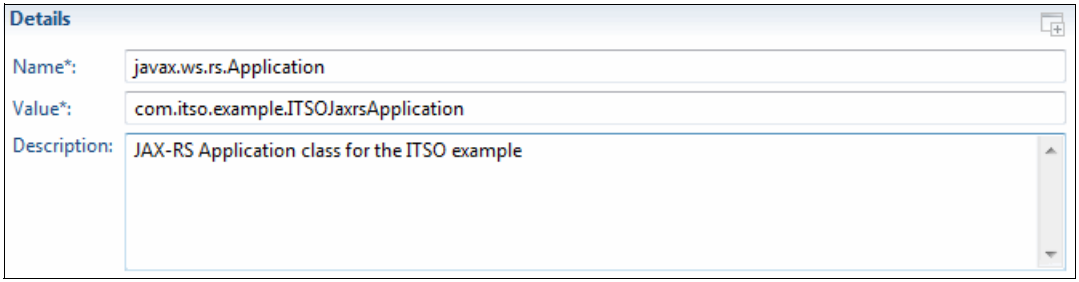


Figure 3-19 Creating an initialization parameter for the JAX-RS servlet

Test your application

Open a web browser and visit:

`http://localhost:9080/ITSOWeb/jaxrs/example`

You should see the output `Hello ITSO` from JAX-RS.

Note: When you access web applications with JAX-RS enabled, you might see the error, shown in Example 3-1, displayed in the console output. This message does not represent a functional error and can be safely ignored.

Example 3-1 Error message with JAX-RS enabled

```
[ERROR   ] An exception occurred during processing of the
org.apache.wink.server.internal.providers.exception.EJBAccessExcepti
onMapper class. This class is ignored.
```

3.1.5 Debugging applications with the Liberty profile developer tools

One of the most powerful features of developing in the Liberty profile developer tools environment is the ability to debug code running on the server directly from the development environment. The following procedure is an example of how you debug a simple application in the Liberty profile developer tools environment.

In order to debug applications, complete the following series of steps:

1. The server must be run in debug mode. You can start (or restart) the server in debug mode by clicking the **bug** icon in the Servers view, as shown in Figure 3-20.

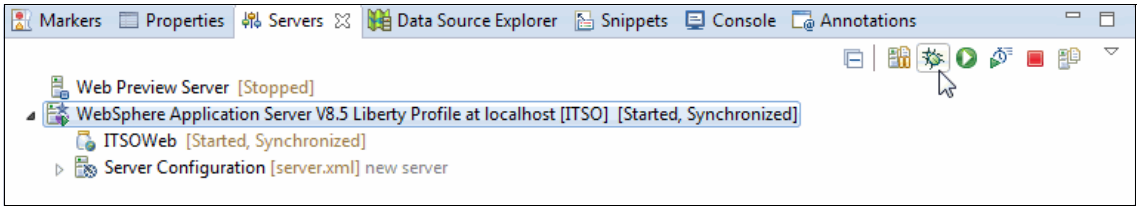


Figure 3-20 Restarting the server in debug mode

2. To debug the application, set a breakpoint in the **`doGet`** method of the `HelloITSO` servlet. On most platforms you can do this by selecting a line in the method and pressing CTRL+SHIFT+b. You can also right-click the vertical bar to the left of the line of code and select **Toggle Breakpoint**.
3. Open a web browser and enter the address for the `HelloITSO` example:

   `http://localhost:9080/ITSOWeb/HelloITSO`
4. A dialog in the tools will ask you if you want to switch to the debug perspective (unless you have already configured eclipse to switch automatically). Click **YES**, and you will see that the debugger has hit the breakpoint in the **`doGet`**

method. Make a change to the method and save the file. Advance past the breakpoint by selecting **Resume** on the menu bar (or by clicking F8.) Notice that the output in the browser has changed based on the changes you made.

3.2 Developing outside the Liberty developer tools

Developers who prefer working outside of the developer tools environment can also enjoy a simplified development experience by using the Liberty profile server. Liberty profile is designed to reduce development overhead by providing a fast, lightweight environment for application development. The following list covers some of the key design features of the Liberty profile server that contribute to the new environment:

- Quick server startup
- Hot deployment of applications
- Dynamic feature enablement
- Dynamic configuration
- Simple configuration that can be modified in a text editor

These characteristics of the Liberty profile ensure that developers can write, debug, and update applications without wasting time on server restarts or updating configuration through a complex administrative interface.

In this section we cover several considerations to keep in mind when developing applications outside of the Liberty developer tools environment.

3.2.1 Feature enablement

The Liberty profile server will only load into memory the subset of the runtime that is being used. You must specifically enable features in the configuration for them to be available to your applications. The following is a list of features used in the examples in this chapter:

servlet-3.0:	Servlet 3.0
jsp-2.2**:**	JavaServer Pages (JSP) 2.2
jpa-2.0:	Java Persistence Architecture (JPA) 2.0
jaxrs-1.1:	Java API for Restful Web Services (JAX-RS) 1.1

Note that some of those listed features also provide other features. For example, the jsp-2.2 feature contains the servlet-3.0 feature. If your server configuration has jsp-2.2 enabled, it does not need to have servlet-3.0 additionally specified to run servlets.

When you add or remove a feature, the runtime is dynamically updated to enable or disable the function. You can test this by deploying a simple servlet and removing the servlet-3.0 feature from the configuration. When you remove the feature, you will see the text in Example 3-2 in the console output.

Example 3-2 Console output after removing the servlet feature

```
[AUDIT   ] CWWKF0013I: The server removed the following features:
[servlet-3.0].
```

If you attempt to load a servlet, the server should fail to respond to the request. Now, enable the servlet-3.0 feature again. You should see the output in Example 3-3 in the console.

Example 3-3 Console output after enabling the servlet feature

```
[AUDIT   ] CWWKF0012I: The server installed the following features:
[servlet-3.0].
```

If you load the servlet now, the request succeeds.

3.2.2 Dynamic application update

By default, applications are automatically updated whenever a change to the application files is made. This allows you to save time that is not lost restarting applications manually.

This behavior can be seen by updating the `web.xml` file for a deployed Web Archive (WAR). Make a change, such as changing the value of a servlet mapping, and save the file. Then load the servlet using the new mapping, and observe that the application is automatically updated.

3.2.3 Common development configuration

Liberty profile uses reasonable default values for configuration whenever possible. Some common cases where you might decide to update the configuration during development are covered next in this section.

Context root for web applications

Unless it has been explicitly defined in the `server.xml` file or in the application, the context root for a web application will be automatically determined from the file name of the WAR file. By default, the value of the context root is the WAR file name without the WAR extension. If you want to change the value of the context

root, it can be specified in the configuration using the application definition shown in Example 3-4.

Example 3-4 An application definition from server.xml

```
<application location="ITSOWeb.war" context-root="myWebAppRoot"/>
```

HTTP server ports

The default port for HTTP requests is 9080. You can also change this value in the configuration. To change the port to 9090, add the endpoint definition in Example 3-5 to the `server.xml` file.

Example 3-5 An endpoint definition from server.xml

```
<httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9090"/>
```

3.2.4 Dynamic configuration

Dynamic configuration updates in Liberty profile server mean that you do not have to waste time restarting the server when you make changes to the configuration. This section covers some scenarios that display the benefits of dynamic configuration updates.

Changing the HTTP server port

If port 9080 is in use on your machine or you are running multiple server instances, you might want to change the default HTTP port. With the server running, add the endpoint configuration, shown in Example 3-5, to your `server.xml` file. Save the file, and try to access a web application from the updated port. Note that the server did not need to restart to make this change.

Add a JNDI entry

Liberty profile server gives you the ability to bind JNDI entries in the configuration. These values are automatically updated when they are added, updated, or removed from the configuration.

To test this, create a simple servlet and add code similar to that shown in Figure 3-21 on page 64.

```
try {
    Context ctx = new InitialContext();
    String value = (String) ctx.lookup("jndi/exampleValue");
    pw.println("<BR>Example value: " + value);
} catch (NamingException ex ) {
    pw.println("Uh oh. I can't find the example value.");
}
```

Figure 3-21 Look up a value using JNDI

Run the servlet, and notice that the servlet was unable to find the value. Now, add the XML element in Example 3-6 to the `server.xml` file.

Example 3-6 XML element in server.xml file

```
<jndiEntry value="This is a sample value"
jndiName="jndi/exampleValue"/>
```

Save the file and without restarting the server, run the servlet again. Notice it now prints out the value that was set in the server configuration. You can also test updating the value or removing the `jndiEntry` element. In each case, the server will be dynamically updated without a restart.

Caution: Only the server configuration is dynamically updated. Values are only read from `bootstrap.properties` when the server starts, so any value changes will not take place until you restart the server.

3.2.5 API JARs

API JARs for technologies included in the Liberty profile server are located in the `dev` directory under the Liberty profile server home directory. The following list notes those directories:

- JavaEE APIs are located in `dev/spec`.
- IBM APIs are located in `dev/ibm-api`.
- Third-party APIs are located in `dev/third-party`.

The following list shows the API JARs used for technologies covered in this chapter:

- Servlets

  `dev/spec/com.ibm.ws.javaee.servlet.3.0_1.0.0.jar`

- JSP

  `dev/spec/com.ibm.ws.javaee.jsp.2.2_1.0.0.jar`

- JSF

  dev/spec/com.ibm.ws.javaee.jsf.2.0_1.0.0.jar
  dev/spec/com.ibm.ws.javaee.jsf.tld.jar
  dev/spec/com.ibm.ws.javaee.jstl.1.2_1.0.0.jar

- JAX-RS

  dev/spec/com.ibm.ws.javaee.jaxrs.1.1_1.0.0.jar
  dev/ibm-api/com.ibm.websphere.appserver.api.jaxrs_1.0.0.jar
  dev/third-party/com.ibm.websphere.appserver.thirdparty.jaxrs_1.0.0.jar

Caution: Only `spec` and `ibm-api` libraries are visible to applications by default. See 3.3, “Controlling class visibility in applications” on page 69 for more information about class visibility.

3.2.6 Debugging applications

You can use a Java debugger to debug your applications by starting the server in debug mode. To do this, start the server with the command **`server debug`**. The server will wait for a Java debugger to attach on port 7777 before starting up. Optionally, you can change the port that the server uses for debugging by setting the value of `WLP_DEBUG_ADDRESS` to the desired port.

3.2.7 Using Maven to automate tasks for the Liberty profile

Apache Maven is a build scripting tool designed to hide some of the complexities of build management from the user. Maven simplifies dependency management by maintaining a set of versioned artifacts in a repository.

Liberty profile enables building with Maven by providing a plug-in that can be used for common administrative tasks and a repository that contains certain API JARs.

Plug-in repository

In order to use Maven for automating development tasks, first include the Liberty profile plug-in, in your project’s `pom.xml` file. The plug-in is located in a public central repository. Example 3-7 shows a sample plug-in definition.

Example 3-7 Maven plug-in repository definition

```
<pluginRepository>
    <id>Liberty</id>
    <name>Liberty Repository</name>
```

```
<url>http://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasd
ev/maven/repository/</url>
    <layout>default</layout>
    <snapshots>
        <enabled>false</enabled>
    </snapshots>
    <releases>
        <enabled>true</enabled>
    </releases>
  </pluginRepository>
```

Creating a server

Use the following command to create a server named SERVER_NAME in the directory SERVER_HOME:

```
mvn liberty:create-server -DserverHome=SERVER_HOME
-DserverName=SERVER_NAME
```

Deploying and undeploying applications

Use the following command to deploy an application located at APPLICATION_FILE to the server named SERVER_NAME:

```
mvn liberty:deploy -DserverName=SERVER_NAME -DserverHome=SERVER_HOME
-DappArchive=APPLICATION_FILE
```

The server must be started to deploy the application.

Use the following command to undeploy the same application: mvn liberty:undeploy -DserverName=SERVER_NAME -DserverHome=SERVER_HOME -DappArchive=APPLICATION_NAME

The server must be started to remove the application. Note that in this scenario, the appArchive parameter uses the application name rather than a file path.

Starting and stopping the server

Use the following command line to start a Liberty profile server instance:

```
mvn liberty:start-server -DserverHome=SERVER_HOME
-DserverName=SERVER_NAME
```

Use the following command to stop a running server:

```
mvn liberty:stop-server -DserverHome=SERVER_HOME
-DserverName=SERVER_NAME
```

Packaging and installing the server

Use the following command line to package an existing server named SERVER_NAME in a file named ARCHIVE_FILE:

```
mvn liberty:package-server -DserverName=SERVER_NAME
-DserverHome=SERVER_HOME -DpackageFile=ARCHIVE_FILE
```

Use the following command to install a server from an archive named ARCHIVE_FILE:

```
mvn liberty:install-server -DassemblyArchive=ARCHIVE_FILE
```

Dependency repositories

The API JARs that are located in the dev/ibm-api directory underneath the Liberty Profile Server home directory are also available in a public maven repository. To use the repository, add the repository definition in Example 3-8 to your pom.xml file.

Example 3-8 Maven repository definition

```
<repository>
   <id>wlp.central</id>

<url>http://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasd
ev/maven/repository</url>
<releases>
   <enabled>true</enabled>
</releases>
<snapshots>
   <enabled>false</enabled>
</snapshots>
</repository>
```

Example 3-9 provides an example of referencing a specific artifact dependency. In this scenario, the example is referencing the IBM JAX-RS extension APIs.

Example 3-9 IBM JAX-RS extension APIs

```
<dependency>
   <groupId>com.ibm.websphere.appserver.api</groupId>
   <artifactId>com.ibm.websphere.appserver.api.jaxrs</artifactId>
   <version>1.0</version>
   <scope>provided</scope>
</dependency>
```

JavaEE dependencies, such as the servlet API, are not available in the IBM repository. These dependencies can be accessed using the repository definition in Example 3-10.

Example 3-10 Repository definition

```
<repository>
   <id>java.net</id>
   <url>http://download.java.net/maven/2</url>
</repository>
```

The dependency definition in Example 3-11 shows how to reference the JavaEE 6 API.

Example 3-11 JavaEE 6 API reference

```
<dependency>
   <groupId>javax</groupId>
   <artifactId>javaee-api</artifactId>
   <version>6.0</version>
   <scope>provided</scope>
</dependency>
```

3.2.8 Using ANT to automate tasks for the Liberty profile

Liberty profile provides an ANT plug-in so you can automate build processes that include management of the server and applications.

ANT plug-in

The Liberty profile ANT plug-in is located in the `dev/tools/ant` directory under the server home directory. To use the plug-in, you must make it available on the classpath. To make the plug-in available, copy the plug-in jar file to the `lib` directory of the ANT installation and reference it using an `antlib namespace`, as shown in Example 3-12.

Example 3-12 Using antlib to include the ANT plug-in

```
<project .... xmlns:wlp="antlib:com.ibm.websphere.wlp.ant">

    ...

  </project>
```

Deploying and undeploying applications

You can automate the addition and removal of applications using the deploy and undeploy tasks shown in Example 3-13.

Example 3-13 Example of deploy and undeploy tasks

```
  <wlp:deploy file="${basedir}/resources/ITSOExample.ear"
timeout="40000"/>

<wlp:undeploy file="ITSOExample.ear" timeout="60000" />
```

Server administration

The following tasks are available for automating server administration tasks:

- Create

  ```
  <wlp:server installDir=”${wlp_install_dir} operation=”create”/>
  ```

- Start

  ```
  <wlp:server installDir="${wlp_install_dir}" operation="start"
  serverName="${serverName}" />
  ```

- Stop

  ```
  <wlp:server installDir=”${wlp_install_dir}” operation=”stop”
  serverName=”${serverName}” />
  ```

- Status

  ```
  <wlp:server installDir=${wlp_install_dir}” operation="status"/>
  ```

- Package

  ```
  <wlp:server installDir=”${wlp_install_dir}” operation=”package”
  archive=”packagedServer.zip”/>
  ```

3.3 Controlling class visibility in applications

Class visibility in the Liberty profile server is controlled by configuring a classloader for an application in the server configuration. Defining the classloader allows you to share common libraries with other applications and control the types of APIs that are visible to the application. Defining the classloader also ensures that applications load classes from their own libraries before using classes from the Liberty profile server runtime.

3.3.1 Using shared libraries in applications

JavaEE application development often involves using common utility JARs across several applications. You can avoid having to maintain several copies of the same utility JARs by using shared libraries.

A library, in the Liberty profile server, is composed of a collection of filesets that reference the JAR files to be included in the library. The library can be used in an application by creating a classloader definition for the application in the server configuration. The application classloader can share the in-memory copy of the library classes with other applications, or it can have its own private copy of the classes loaded from the same location.

Shared libraries, defined in the configuration, can only be used by applications that are explicitly configured in the `server.xml` file. Applications without configuration, such as those placed in the `dropins` directory, can use a global shared library as discussed in 3.3.5, “Global libraries” on page 72.

3.3.2 Creating a shared library in the Liberty profile developer tools

Use the following procedure to create a shared library in the Liberty profile developer tools:

1. Load the `server.xml` file in the design editor.
2. Right-click the application and click **Add** → **Classloader Service**.
3. In the Classloader Service properties, on the resulting panel, click **New** to enter either the library references or shared library references.
4. Right-click **Classloader Service** and then click **Add** → **Fileset**.
5. Fill in the properties for the Fileset and save the `server.xml` file.

Look at the source view for the `server.xml` file. Notice that the application now contains a classloader, which contains a library, which contains a fileset. The Liberty profile configuration is flexible enough to allow many configuration elements to be specified either as a child of a parent element, or as an attribute reference from that parent element. The shared library configuration in 3.3.3, “Creating a shared library outside of the tools” on page 71 displays how some of the same configuration can be specified using references.

Caution: The Liberty profile developer tools use slightly different terminology than the server configuration file. A library in the tools is equivalent to a `privateLibrary` in `server.xml`. A shared library in the tools is equivalent to a `commonLibrary` in `server.xml`.

3.3.3 Creating a shared library outside of the tools

To create a shared library outside of the Liberty profile developer tools, you need to manually edit the `server.xml` file to add the following:

- A fileset that references the JAR files on disk.
- A library that either contains or references the fileset.
- A classloader for each application that uses the shared library. The classloader should reference the library element. If you want the application to share the in-memory copy of the library, use the attribute `commonLibraryRef` to reference the library. If you want the application to have its own copy of the library, use the attribute `privateLibraryRef`.

Example 3-14 shows the definition of a shared library used by two different applications. Both applications will share an in-memory copy of the classes.

Example 3-14 Definition of two applications using the same shared library

```
<library id="loggingLibrary">
   <fileset dir="${server.config.dir}/logger"
includes="ITSOLogger.jar"/>
</library>

<application name="ITSOApp1" location="ITSO_1.ear">
   <classloader commonLibraryRef="loggingLibrary"/>
</application>

<application name="ITSOApp2" location="ITSO_2.ear">
   <classloader commonLibraryRef="loggingLibrary"/>
</application>
```

3.3.4 Using libraries to override Liberty profile server classes

In some situations, you might decide to override certain classes in the Liberty profile server with your own version of the same classes. You can accomplish this by creating a shared library that contains the classes you want to use. Use the following options:

- In the Liberty profile tools

  On the classloader properties panel, change the value in the drop-down for delegation to parentLast.

- Outside the Liberty profile tools

  To change the classloader delegation outside the Liberty profile developer tools, add the attribute `delegation="parentLast"` on the `classloader` element in the `server.xml` file.

3.3.5 Global libraries

The Liberty profile server allows you to provide a library or set of libraries that can be accessed by all applications. To add a JAR file to this global shared library, simply copy it to one of the following two locations under `${WLP_USER_DIR}`:

- `shared/config/lib/global`
- `servers/`*`server_name`*`/lib/global`

The libraries in these directories will be used by all applications that do not specifically define a classloader. If you define a classloader for an application, the global library classes will not be available unless you add the library `global` as a shared library. The sample configuration in Example 3-15 shows an application that has been configured to use the global shared library.

Example 3-15 Application using a global shared library

```
<application name="ITSOApp1" location="ITSO_1.ear">
   <classloader commonLibraryRef="global"/>
</application>
```

3.3.6 Using a classloader to control API visibility

By default, an application in Liberty profile can see supported specification APIs (such as the Java EE Servlet specification) and IBM-provided APIs. For the application to load classes from third-party libraries, such as those located in `dev/third-party`, you need to configure the allowed API types for the application classloader. In the Liberty profile developer tools, you can configure this value in the Allowed API types field on the classloader service details panel. If you are directly editing the `server.xml` file, add the attribute `apiTypeVisibility` to the `classloader` element. In both cases, the value is to be a comma-separated list of any combination of the following:

`spec`	Supported Specification Libraries
`ibm-api`	IBM-provided APIs
`ibm-spi`	IBM-provided SPIs
`third-party`	Third-party APIs

For example, the following classloader element allows an application to see specification libraries, IBM-provided APIs, and third-party APIs:

```
<classloader apiTypeVisibility="spec,ibm-api, third-party"/>
```

4

Iterative development of OSGi applications

In addition to Java EE applications, WebSphere Application Server Liberty Profile allows you to develop OSGi applications. OSGi provides a framework for developing flexible, modular applications. With OSGi applications, you can deploy and manage applications as a set of versioned OSGi bundles.

The chapter contains the following sections:

- Introduction to OSGi applications in Liberty profile
- Developing OSGi applications in the Liberty profile developer tools
- Developing OSGi applications outside the Liberty profile developer tools

4.1 Introduction to OSGi applications in Liberty profile

The Liberty profile server and Liberty profile developer tools help you to develop and deploy modular OSGi applications quickly. The advantages of OSGi are included in the following list:

- Reduces complexity through modular design.
- Integrates with standard JavaEE technologies such as Servlets.
- Promotes service oriented design.
- Composes isolated enterprise applications using multiple versioned bundles with dynamic life cycles.
- Provides careful dependency and version management.
- Offers declarative assembly of components.
- Allows sharing modules through a bundle repository that can host common and versioned bundles.
- Allows access to external bundle repositories.
- Allows administrative updates to deployed applications at a bundle level.

For more information, refer to the OSGi home page at:

http://www.osgi.org

The OSGi support in the WebSphere Application Server Liberty profile is based on the Apache Aries open community project.

4.2 Developing OSGi applications in the Liberty profile developer tools

The Liberty profile developer tools make developing OSGi applications easier by providing graphical editors for OSGi artifacts such as OSGi Blueprint and manifest files. The tooling environment allows you to safely create dependencies on other bundles without having to worry about problems being introduced by mistakes in manual editing of manifest files. The Liberty profile developer tools can also assist in converting existing Java EE applications into OSGi applications.

4.2.1 Using the tools to build an OSGi application

In this section, we describe how to build a simple hello world style OSGi application using different bundles for client, service, and API. The application will show a limited subset of what is possible with OSGi applications, and will show how to get started building OSGi applications in the Liberty profile developer tools.

Create OSGi bundle projects

In this application, we will use three bundles: a client bundle, a service bundle, and an API bundle. Begin by creating a project for each bundle using the following process:

1. Click **File** → **New** → **OSGi Bundle Project**.
2. Enter a project name. This example uses the project names `ITSO.OSGIExample.API`, `ITSO.OSGiExample.Client`, and `ITSO.OSGiExample.Service`.
3. For the Target Runtime, select **WebSphere Application Server V8.5 Liberty Profile**.
4. Uncheck the **Add bundle to application** check box. If you leave this selected, a new OSGi application project will be created.
5. Leave all other values as the default settings and exit the wizard by selecting **Next** → **Next** → **Finish**.

Create the client API

In the `ITSO.OSGiExample.Client` project, create an interface named `HelloAPI` in the com.itso.example.osgi.api package. The interface should contain a single public method with a void return type named `sayHello`.

Expose this interface class to other bundles by editing the manifest file for the project. You can open the manifest editor by double-clicking the Manifest element underneath the bundle project, as shown in Figure 4-1 on page 78.

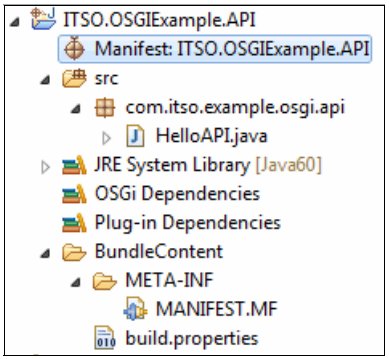


Figure 4-1 API bundle project

In the manifest editor, click the **Runtime** tab. In the exported packages pane, click **Add**. Select the `com.itso.example.osgi.api` package from the list and then click **OK**.

Create the service bundle

The OSGi service bundle will provide an implementation of the API. For the API's interface to be visible in the service project, import the API package in the service bundle. To import the API package, complete the following steps:

1. Open the manifest editor for the `ITSO.OSGiExample.Service` project. In the Imported Packages pane of the Dependencies tab, click **Add**. In the package selection dialogue, select `com.itso.example.osgi.api`. Then, save and close the manifest.
2. Create an implementation for the API by creating a new Java class in the `ITSO.OSGiExample.Service` project named com.itso.example.osgi.service.HelloImpl. The class should implement the `HelloAPI` interface. In the implementation of the `sayHello()` method, enter an output statement such as `System.out.println("Hello, Liberty developers from OSGi").` Save and close the file.
3. Export the service implementation using an OSGi Blueprint file. Begin by creating the file. Right-click the **`ITSO.OSGiExample.Service`** project and select **New** → **Blueprint File**. Leave all values as the defaults, and click **Finish**.
4. In the Blueprint editor's design tab, click **Add**. Select **Bean** and click **OK**. Click **Browse** and select **`HelloImpl`**. Leave the Bean ID field as the default value, `HelloImplBean`.
5. Select **Blueprint** in the Overview pane and click **Add**. Select **Service**, and click **OK**. Click **Browse** next to the Service Interface field and select **HelloAPI.** Click **Browse** next to the Bean Reference field and select

HelloImplBean. Click **OK** and save the file. Examine the source for the Blueprint file, and verify that it resembles the XML shown in Figure 4-2.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
    <service id="HelloImplBeanService" ref="HelloImplBean"
        interface="com.itso.example.osgi.api.HelloAPI" />
    <bean id="HelloImplBean"
        class="com.itso.example.osgi.server.HelloImpl" />
</blueprint>
```

Figure 4-2 Service Blueprint file

Create an OSGi client bundle

The OSGi client bundle calls the `sayHello` method on the service implementation. Similar to the service bundle, the client bundle must have access to the API by importing the API package in its manifest. Use the procedure from “Create the service bundle” on page 78 to edit the manifest and import the API.

Create a `HelloITSOClient` class in the `ITSO.OSGiExample.Client` project. The class should contain the following pieces:

- A private field of type `HelloAPI` named `helloService`
- Get and Set methods for the `helloService` field
- A method named `init` that will be called when the client is initialized. This method should print out a message and then execute `helloService.sayHello()`.

The completed class should resemble the code shown in Figure 4-3 on page 80.

```
package com.itso.example.osgi.client;

import com.itso.example.osgi.api.HelloAPI;

public class HelloITSOClient {

    private HelloAPI helloService = null;

    public void init() {
        System.out.println("The client is starting.");
        helloService.sayHello();
        System.out.println("The client is finished initializing.");
    }

    public HelloAPI getHelloService() {
        return helloService;
    }

    public void setHelloService(HelloAPI helloService) {
        this.helloService = helloService;
    }


}
```

Figure 4-3 Source code for HelloITSOClient

The client must be configured so that it can recognize the service implementation and have the service dependency injected into the helloService field. This is done by creating a Blueprint file for the client using the following steps:

1. Create a new Blueprint file by right-clicking the **`ITSO.OSGiExample.Client`** project and selecting **New** → **Blueprint File**. You can accept all default values for the file and click **Finish**.
2. Add a reference element to the Blueprint by clicking **Add** on the Overview pane in the Blueprint editor. Select **Reference** and click **OK**. Click **Browse** beside the Reference Interface field, and select **`HelloAPI`** from the list of classes. In the Reference ID field, enter `helloRef`. Click **OK** to finish adding the reference element.
3. Add a bean element to the Blueprint file by again clicking **Add** in the Overview pane. Select **Bean** and click **OK**. Click **Browse**, select **HelloITSOClient**, and click **OK**. Click **OK** again to finish adding the bean element. In the bean properties panel on the right side of the editor, locate the Initialization Method field and click **Browse**. Select the **`init()`** method and click **OK**. This will alert the Blueprint container to run the `init` method when the `HelloITSOClient` class is instantiated.
4. Add a property to the HelloITSOClientBean by selecting the bean in the Overview pane and clicking **Add**. Select **Property** and click **OK**. In the Details panel, enter `helloService` in the Name field. Then click **Browse** beside the

Reference field and select the **helloRef** reference. Click **OK** to finish adding the property.

5. Save the Blueprint file and switch to the Source tab.

The completed file should resemble Figure 4-4.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
    <reference id="helloRef"
        interface="com.itso.example.osgi.api.HelloAPI" />
    <bean id="HelloITSOClientBean"
        class="com.itso.example.osgi.client.HelloITSOClient"
        init-method="init">
        <property name="helloService" ref="helloRef"/>
    </bean>
</blueprint>
```

Figure 4-4 Blueprint file for ITSO.OSGiExample.Client

Create an OSGi application

Create an OSGi application to contain the three OSGi bundles by selecting **File** → **New** → **OSGi Application Project**. Enter a project name such as `ITSO.OSGiExample.App`. Leave the other values as the defaults and click **Next**.

In the contained bundles list, select all three OSGi bundle projects, and click **Finish**.

4.2.2 Using the tools to deploy and test an OSGi application

To deploy the OSGi application to a WebSphere Liberty profile server instance, right-click the server in the Servers view and select **Add and Remove**. In the Add and Remove panel, select the **`ITSO.OSGiExample.App`** application and click **Add**. Then click **Finish** to complete adding the application to the server.

The application will now be deployed and started. You should see output in the console similar to Figure 4-5.

```
[AUDIT   ] CWWKF0012I: The server installed the following features: [blueprint-1.0].
[AUDIT   ] CWWKF0008I: Feature update completed in 1.451 seconds.
[AUDIT   ] CWWKG0017I: The server configuration was successfully updated in 2.433 seconds.
[AUDIT   ] CWWKZ0001I: Application ITSO.OSGIExample.App started in 1.778 seconds.
The client is starting.
Hello, Liberty developer.
The client is finished initializing.
```

Figure 4-5 Output after adding the OSGI application to the server

Note that the blueprint-1.0 feature is added automatically. The client's `init` method will be called during the application start, so you will see the messages printed out in the `init` method in the console. Note also that the message from the service implementation is printed out because the client has invoked the `sayHello` method.

4.2.3 Adding an OSGi web application bundle

Now add an OSGi web application bundle to the OSGi application. The web application will make use of the existing service inside a servlet. Rather than using Blueprint to manage the service reference, use the service registry to locate the service. Complete the following steps:

1. Begin by creating a new OSGi bundle project. This process is similar to the process described in "Create OSGi bundle projects" on page 77. The following list shows the differences:
   a. On the first panel of the create project wizard, select the **Add Web Support** check box.
   b. Leave the check box for adding the bundle to the OSGi application checked so that it will be added to the `ITSO.OSGiExample.App` application.
   c. Optionally, on the web module panel, enter a different value for the context root. In this example, we use a context root of `osgi`.
2. As was the case with the service and client bundles, you need to edit the manifest for the web application bundle to import its dependencies. Open the manifest editor and navigate to the Dependencies tab. Notice that several servlet dependencies were automatically added to the manifest when the project was created. Add the `com.itso.example.osgi.api` package to the list of dependencies. Additionally, you need to add `org.osgi.framework` version 1.5.0 to the list.
3. Create a servlet in the new bundle project using the Create Servlet wizard. Name the servlet `HelloWABServlet`, and edit the servlet mapping value to be `hello`. For more information about creating servlets in the Liberty profile developer tools, see 3.1.1, "Using the tools to create a simple servlet application" on page 44.
4. In the servlet, add a private field of type `HelloAPI` named `helloService`. In the `doGet` method, add a call to `helloService.sayHello()`.
5. Also in the servlet, override the `init(ServletContext)` method. In this method, you use the `ServletContext` to get the `ServletConfig`. From the `ServletConfig`, get the `osgi-bundlecontext` attribute. This is the OSGi BundleContext object. You can use this BundleContext to look up the `HelloAPI` service in the service registry. The complete `init` method is shown in Figure 4-6 on page 83.

```
@Override
public void init(ServletConfig config) throws ServletException {
    super.init(config);
    ServletContext context = config.getServletContext();
    BundleContext ctx = (BundleContext) context.getAttribute("osgi-bundlecontext");
    ServiceReference ref = ctx.getServiceReference(HelloAPI.class.getName());
    helloService = (HelloAPI) ctx.getService(ref);
}
```

Figure 4-6 Example init method for the HelloWABServlet

6. Save the servlet.

You can test this by visiting:

`http://localhost:9080/osgi/hello`

You will not see any output in the browser (unless you added output in the servlet's `doGet` method) but you will see the message `Hello, Liberty developer` displayed in the console output. This message shows that the servlet successfully retrieved the service from the service register in its `init` method and invoked the service's `sayHello` method in its `doGet` method.

4.3 Developing OSGi applications outside the Liberty profile developer tools

Developers who prefer working outside of the Liberty profile developer tools environment can still develop OSGi applications quickly using the Liberty profile server. In this section, we discuss some considerations for developing in this environment and some tools to automate build and deployment processes.

4.3.1 Building and deploying OSGi applications outside of the tools

When building OSGi applications outside the Liberty profile developer tools, note that there are several API jars located in the `dev/spec` directory that contain important core classes such as BundleContext and ServiceReference:

- `com.ibm.ws.org.apache.aries.blueprint.1.0.0_1.1.0.jar`
- `com.ibm.ws.org.apache.aries.blueprint_0.4.1.ibm-s20120308-0347_1.0.1.jar`
- `com.ibm.ws.org.osgi.cmpn.4.2.0_1.0.0.jar`
- `com.ibm.ws.org.osgi.core.4.2.0_1.0.0.jar`
- `com.ibm.ws.org.osgi.service.obr.1.0.2_1.0.0.jar`

Enterprise bundle archive (EBA) files can be placed in the dropins directory, such as EAR or WAR files. You can also define them explicitly using an application element in the `server.xml` file.

Also note that you might need to manually enable the features `blueprint-1.0` and `wab-1.0` in the server configuration.

4.3.2 Using Maven to automate OSGi development tasks

You can automate the building and deployment of OSGi applications by using the Liberty Maven plug-in, in conjunction with open source tools. An example of an open source tool is the EBA Maven plug-in developed by Apache Aries.

As discussed in 3.2.7, “Using Maven to automate tasks for the Liberty profile” on page 65, you can use the Liberty Maven plug-in to automate tasks. Some examples of these tasks are deploying applications and starting a Liberty profile server.

The EBA Maven plug-in allows you to build EBA archives from component bundles. It can automatically generate an Application Manifest file from information contained in the Maven `pom` file. For more information about the EBA Maven plug-in, see:

`http://aries.apache.org/modules/ebamavenpluginproject.html`

4.3.3 Using ANT to automate OSGi development tasks

With Apache Ant and the Liberty Ant plug-in, you can automate the packaging and deployment of OSGi applications. Refer to 3.2.8, “Using ANT to automate tasks for the Liberty profile” on page 68 for more information about using the Liberty plug-in. This section gives further details on how to automate tasks such as deploying and undeploying applications or starting and stopping servers.

You can use standard Ant tasks, such as `zip`, to package OSGi applications that can then be deployed using the Liberty plug-in. Example 4-1 on page 85 shows sample syntax for creating an OSGi application named `ITSOExampleApp`.

Example 4-1 Syntax for creating an OSGi application

```
<zip destfile="${output.dir}/ITSOExampleApp.eba" basedir="${basedir}">
    <filename name="META-INF/APPLICATION.MF"/>
    <fileset dir="${basedir}">
        <include name="*.jar"/>
    </fileset>
</zip>
```

5

Data access in the Liberty profile

The Liberty profile server supports standard Java EE technologies for data access such as the Java Persistence API (JPA) and JDBC. The Liberty profile developer tools simplify the development of data access applications for these technologies by assisting with the setup of required resources. In this chapter, we give an overview of data access in Liberty profile and give examples of how to develop data access applications with the Liberty profile.

The following topics are addressed in this chapter:

- Accessing data using a data source and JDBC
- Developing JPA applications

5.1 Accessing data using a data source and JDBC

JDBC provides a database-independent API for querying and updating data. Liberty profile provides support for data access using JDBC 4.0.

5.1.1 Basic concepts for configuring data access in Liberty profile server

In order to use JDBC in an application, first configure access to the database. In Liberty profile, define a data source that has knowledge of the database. Several key configuration concepts important to this process are described in this section.

Filesets

A fileset is simply a collection of files. They have several purposes in Liberty profile. In the context of data access, they are generally used to point to the JDBC driver library files provided by a database vendor.

Shared libraries

As described in 3.3, “Controlling class visibility in applications” on page 69, shared libraries provide a collection of classes that can be used by one or more applications or data sources. Libraries are composed of fileset elements that define the location of the class files.

JDBC drivers

A JDBC driver definition references a shared library. You can optionally specify individual classes from the vendor-provided JDBC driver library to be used as the data source implementation. If you do not specify the implementation classes, the Liberty profile server will infer the appropriate class name for most common databases.

Data sources

A data source is the key piece of configuration for data access in the Liberty profile server. In the simplest case, a data source is composed only of a JDBC driver and the JNDI name where the data source will be made available. In more complex scenarios, you can specify other properties such as the type of data source, transactional behavior, or vendor-specific database properties. For more information about the various configuration options available for data sources, see the following information center website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/twlp_dep_configuring_ds.html

5.1.2 Adding a data source using Liberty profile developer tools

The WebSphere Liberty profile developer tools facilitate the definition of adding a data source and related entities. The following scenario shows one way to define all of the artifacts necessary to add a data source to your application. Complete the following steps to add a data source:

1. Open the `server.xml` file for your WebSphere Liberty profile server. In the design view, select **Server Configuration** and click **Add**. Select **Data Source** and click **OK** (Figure 5-1).

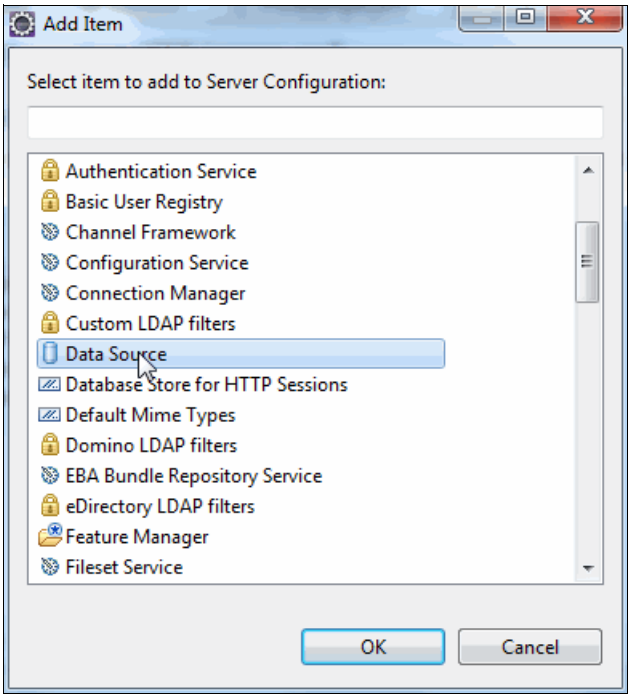


Figure 5-1 Adding a data source to an application

2. In the input field for JNDI Name enter `jdbc/ITSO`.
3. The Data Source requires a JDBC Driver. On the Data Source Details panel, beside JDBC driver, click **New**. This opens a panel to configure the new JDBC driver, as shown in Figure 5-2 on page 90.

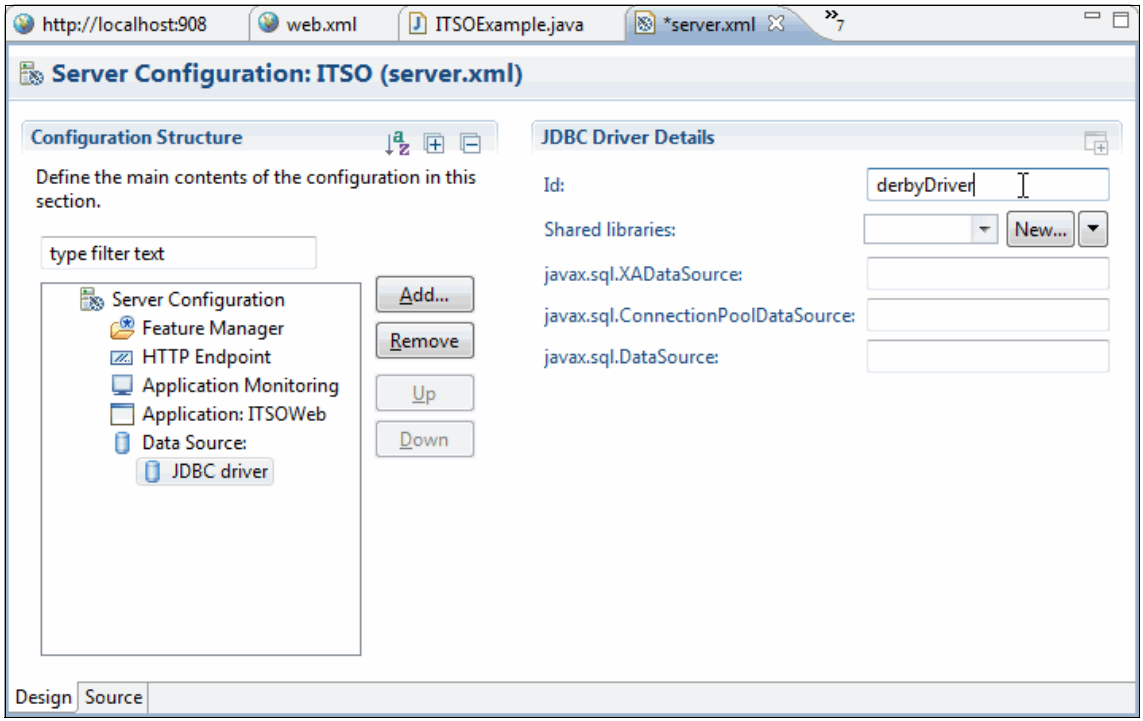


Figure 5-2 JDBC driver configuration

4. A JDBC driver requires a shared library that references the JDBC driver implementation classes. Beside the Shared libraries field, click **New** to create a new shared library. This opens a panel to configure a shared library, as shown in Figure 5-3 on page 91.

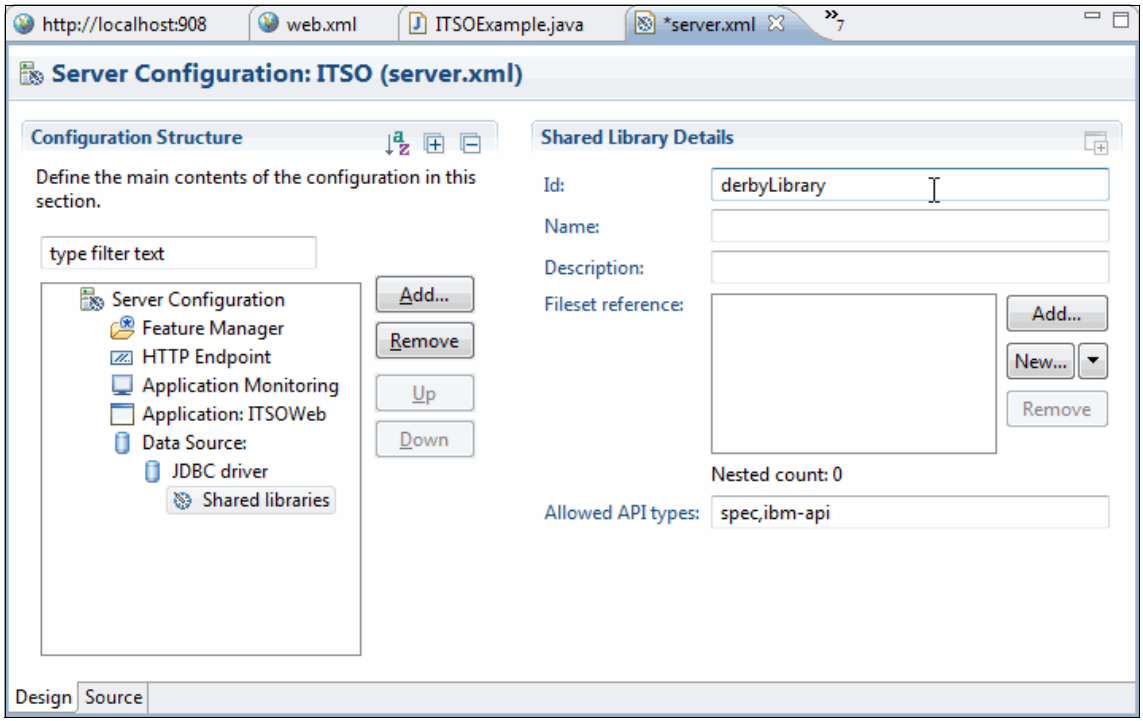


Figure 5-3 Adding a shared library

5. Now, create a new fileset for the shared library by clicking **New** next to the input field for Fileset reference. This opens the Fileset Service Details panel.
6. Enter the directory where the JAR files for your JDBC driver are located in the Base directory field, or click **Browse** to select a directory, as shown in Figure 5-4 on page 92.

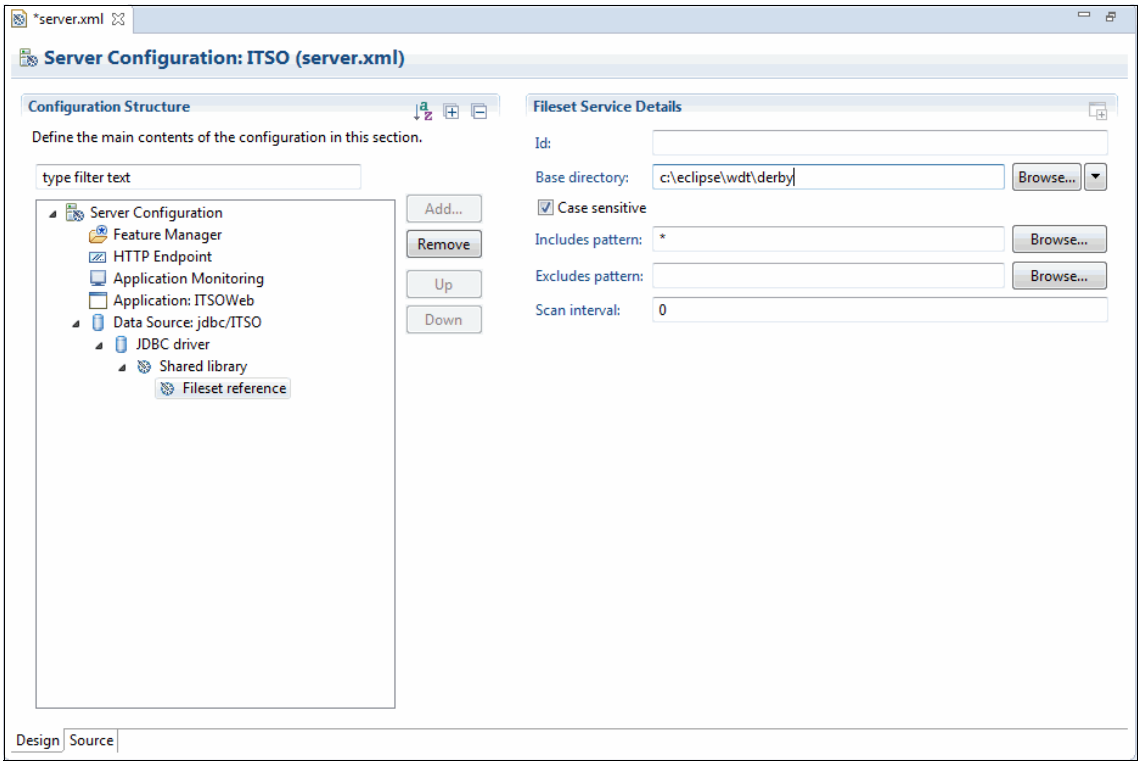


Figure 5-4 Fileset Service Details panel

7. Right-click the **Data Source element** and select **Add** → **Properties for Derby Embedded**.
8. In the Properties panel, select **Create** in the Create database drop-down.
9. Also in the database properties panel, enter `ITSO` in the Database name field.
10. Save the `server.xml` file. The data source is now available for applications.

5.1.3 Adding a data source outside the Liberty profile developer tools

You can also add a data source to the Liberty profile server by simply editing the `server.xml` file in a text editor. The following procedure details the four key parts of creating a data source for your applications:

1. Creating a fileset

   The first step in defining a data source is to create a fileset that contains all of the JAR files required for a JDBC driver. Example 5-1 on page 93 shows a simple fileset definition that will include all files in the directory `c:\Derby`.

Example 5-1 A fileset definition

```
<fileset id="derbyFileset" dir="c:\Derby"/>
```

2. Creating a shared library

   The next step in creating a data source is to create a shared library that contains the fileset or filesets that reference the JDBC driver JAR files. Example 5-2 shows a shared library definition that references the fileset you created earlier. Note that the `filesetRef` attribute is the same as the `id` attribute of the fileset element.

   Example 5-2 Shared library definition

   ```
   <library id="derbyLibrary" filesetRef="derbyFileset"/>
   ```

3. Creating a JDBC driver

   Next, create a JDBC driver that references the shared library you just created. Example 5-3 gives an example JDBC driver definition. Note that the value of the `libraryRef` attribute is the same as the `id` from the library element.

   Example 5-3 JDBC driver definition

   ```
   <jdbcDriver id="derbyDriver" libraryRef="derbyLibrary"/>
   ```

4. Creating a data source

   Finally, create a data source definition. The minimum required configuration for a data source consists of a JDBC driver definition and a JNDI name that your application will use to access the data source. Example 5-4 shows a data source definition that references the JDBC driver you just created.

   Example 5-4 Data source definition

   ```
   <dataSource jdbcDriverRef="derbyDriver" jndiName="jdbc/ITSO"/>
   ```

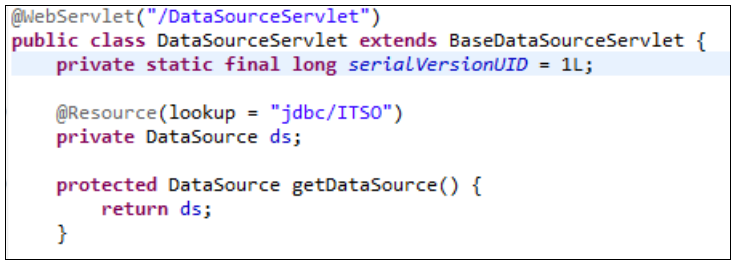
This scenario describes an extremely simple data source configuration. More information about configuring database connectivity in the Liberty profile can be found at the following information center website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/twlp_dep_configuring_ds.html

5.1.4 Using the data source in an application

A data source that is defined in the server configuration can be accessed using JNDI. The `jndiName` property specifies the location of the data source in the namespace. To retrieve the DataSource object, simply look it up in JNDI. The

code in Figure 5-5 displays how to accomplish this using the `javax.annotation.Resource` annotation on a field. You can also obtain the DataSource object by creating a new InitialContext object and invoking the **`lookup`** method with the JNDI name of the data source as an argument.


```
@WebServlet("/DataSourceServlet")
public class DataSourceServlet extends BaseDataSourceServlet {
    private static final long serialVersionUID = 1L;

    @Resource(lookup = "jdbc/ITSO")
    private DataSource ds;

    protected DataSource getDataSource() {
        return ds;
    }
```

Figure 5-5 Retrieving a DataSource in a servlet

Caution: The `javax.annotation.Resource` annotation does not support the **`lookup`** attribute in JDK 1.6. If you are using JDK 1.6, you must change the order of libraries on the build path for the project so that the Liberty library is loaded before the JDK 1.6 library.

5.1.5 Defining a data source in an application

Applications can also define data sources through annotations or a deployment descriptor. To configure a data source this way, the application must have access to the JDBC driver classes. To do this, create a classloader for the application as described in 3.3, “Controlling class visibility in applications” on page 69. The classloader should reference a shared library that contains the JDBC driver classes.

Defining a data source using annotations

Example 5-5 defines a data source in a Java class of an application using annotations. The DataSourceDefinition annotation on the class defines various properties for the data source. Only the name and className are required properties.

Example 5-5 Annotations defining a data source

```
@DataSourceDefinition(name="java:comp/env/jdbc/ITSODataSource",
        className ="org.apache.derby.jdbc.EmbeddedXADataSource40",
        databaseName = "ITSO",
        isolationLevel = Connection.TRANSACTION_READ_COMMITTED,
        loginTimeout = 88,
```

```
        maxPoolSize = 10,
        properties = {"connectionTimeout=0",
                      "createDatabase=create",
                       "queryTimeout=1m20s"})
```

Defining a data source using a deployment descriptor

You can define a data source in a WAR file deployment descriptor by adding a data source element to the `web.xml` file. You can simply edit the file, or use the deployment descriptor editor in the Liberty profile developer tools.

To create the data source in the tools, complete the following steps:

1. Begin by loading the `web.xml` file. In the design view, select the Web Application and click **Add**.
2. On the resulting panel, select **Data Source**. Fill in the properties for the data source. For example, enter `java:comp/env/jdbc/ITSODataSourceDD` for the name, `org.apache.derby.jdbc.EmbeddedDataSource` for the class name, and `ITSODD` for the database name.
3. Now, click the new data source in the panel on the left, and click **Add**. Select **Property** to add a new property. In the property editor, enter `createDatabase` in the name field and `create` in the value field.
4. Save the deployment descriptor. If the server is running, the data source is now available to your applications.

Example 5-6 shows how the completed data source appears in the `web.xml` file.

Example 5-6 Data source definition in a deployment descriptor

```
<data-source>
        <name>java:comp/env/jdbc/ITSODataSourceDD</name>
        <class-name>
                org.apache.derby.jdbc.EmbeddedDataSource
        </class-name>
        <database-name>ITSODD</database-name>
        <property>
                <name>createDatabase</name>
                <value>create</value>
        </property>
   </data-source>
```

The definitions here represent a small portion of the configuration options available for data sources. For more information about application-defined data sources, refer to the following information center website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/rwlp_ds_appdefined.html

5.1.6 Using application-defined data sources

Obtaining an application-defined data source in an application uses the same procedure regardless of whether the data source was defined in a DataSourceDefinition annotation or in a deployment descriptor. In both cases, the data source will be available in the namespace at the location you specified in the name property. Figure 5-6 shows sample code for retrieving a DataSource that was defined in the web application's deployment descriptor.

```
@WebServlet("/WebXmlDataSourceServlet")
public class WebXmlDataSourceServlet extends BaseDataSourceServlet {
    private static final long serialVersionUID = 1L;

    @Resource(lookup = "java:comp/env/jdbc/ITSODataSourceDD")
    DataSource webXmlDataSource;

    @Override
    protected DataSource getDataSource() {
        return webXmlDataSource;
    }
}
```

Figure 5-6 Obtaining a DataSource defined in web.xml

Figure 5-7 shows sample code for accessing a DataSource that was defined in an annotation.

```
@WebServlet("/AnnotationDataSourceServlet")
@DataSourceDefinition(name = "java:comp/env/jdbc/ITSODataSource",
className = "org.apache.derby.jdbc.EmbeddedXADataSource40",
databaseName = "ITSO",
properties = {"createDatabase=create"})
public class AnnotationDataSourceServlet extends BaseDataSourceServlet {
    private static final long serialVersionUID = 1L;

    @Resource(lookup = "java:comp/env/jdbc/ITSODataSource")
    DataSource annotationDataSource;

    @Override
    protected DataSource getDataSource() {
        return annotationDataSource;
    }
```

Figure 5-7 Obtaining a DataSource defined in annotations

5.1.7 Testing the data sources

The additional materials contain an application and three prebuilt Derby databases that can be used to test each data source.

You can use one of the following database options:

- To use the Derby databases, expand `ITSO_Derby.zip`, `ITSO2_Derby.zip`, and `ITSODD_Derby.zip` in the server's `root` directory. The data source definitions in the applications are designed to access these databases.
- To use another database, you need to update the data sources to reference your database. Also, you need to create a table named `USERTBL` in the database with the following configurations:
  - A column named `NAME` with a type that maps to `java.lang.String` (such as `VARCHAR`).
  - A column named `LOCATION` with a type that maps to `java.lang.String`.
  - A column named `ID` with a type that maps to `java.lang.Integer` (such as `INTEGER`). This column should be generated by the database.

The application contains three servlets: DataSourceServlet, AnnotationDataSourceServlet, and WebXmlDataSourceServlet. Each extends the abstract class BaseDataSourceServlet. Each servlet can perform four simple operations on the database table. The operations include selecting all rows, inserting a new row, dropping the table, or creating the table.

Note: Because of database differences, the create table statement might not work for all databases.

Each servlet can be tested using the JSP page `dataSource.jsp` that is included in the additional materials. To access the page, visit:

`http://localhost:9080/ITSOWeb/dataSource.jsp`

5.1.8 Dynamic configuration updates for data sources

Changing the attributes of a data source element at runtime results in dynamic updates to the server. Although the runtime will be updated immediately, some dynamic updates might affect a running server differently than others. For example, changing the isolationLevel attribute will change the isolation level for any future connection request, but current connections will retain their existing isolation level. An update to the value of the commitOrRollbackOnCleanup attribute takes place immediately.

You can find a list of data source properties and their associated configuration update behaviors at:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/rwlp_ds_config_updates.html

The rules noted only apply to data sources that are defined in the server configuration. When you update the properties of an application-defined data source, the entire data source is destroyed and the updated application uses a new data source. Effectively, from a developer perspective, all of the changes will take place immediately.

5.2 Developing JPA applications

The Java Persistence API (JPA) provides a framework for working with relational databases in an object-oriented way. WebSphere Liberty profile server provides support for applications that use application-managed and container-managed JPA written to the JPA 2.0 specification. The support is built on top of Apache OpenJPA with extensions to support the container-managed programming model.

5.2.1 JPA applications in the Liberty profile developer tools

The following procedure allows you to create a simple JPA application in the Liberty profile developer tools:

1. Before beginning, make sure that you have defined a data source as described in 5.1, “Accessing data using a data source and JDBC” on page 88. Your database also needs to be configured with the `USERTBL` as described in 5.1.7, “Testing the data sources” on page 97. The `ITSO_Derby.zip` file in the additional materials provides a preconfigured Derby database.
2. Begin by enabling the JPA-2.0 facet on the ITSOWeb project. This will cause a `persistence.xml` file to be added to the project. If the project is deployed to the Liberty profile server, it will also cause the tools to prompt you to add the JPA-2.0 feature to the server configuration.
3. Create a new JPA entity class by right-clicking the project and selecting **New** → **JPA Entity**. Enter `com.itso.example` in the Java package field and `UserEntity` in the Class name field. Figure 5-8 on page 99 shows the first page of the JPA entity wizard.

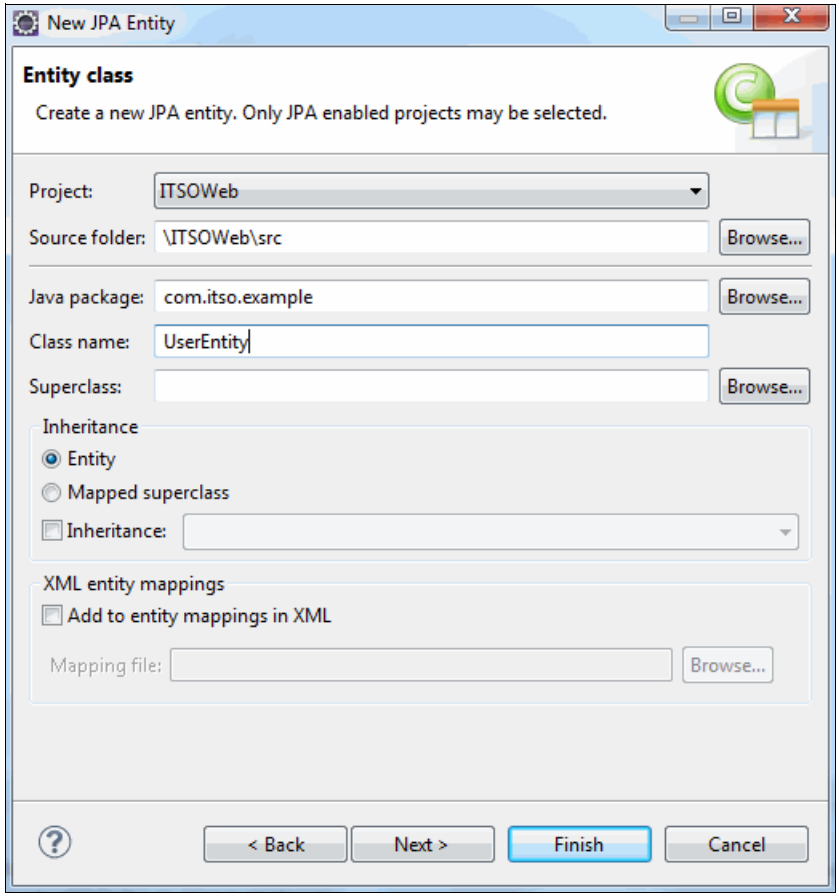


Figure 5-8 New JPA Entity wizard

4. Click **Next** to advance to the next panel. Under Table name, uncheck the **Use default** check box. Enter `USERTBL` in the Table name field. Add an entity field by clicking **Add**. Use the **Browse** button next to the Type field to select `java.lang.String`, enter `location` in the name field, and then click **OK**.
5. Repeat step 4 to add entities for a `java.lang.String` field named `name` and a `java.lang.Integer` field named `id`. Check the **box** in the Key column for the row containing the `id` field to indicate that the field is the primary key for the table. Figure 5-9 on page 100 shows the completed second panel of the JPA wizard. Click **Finish** to exit the wizard.

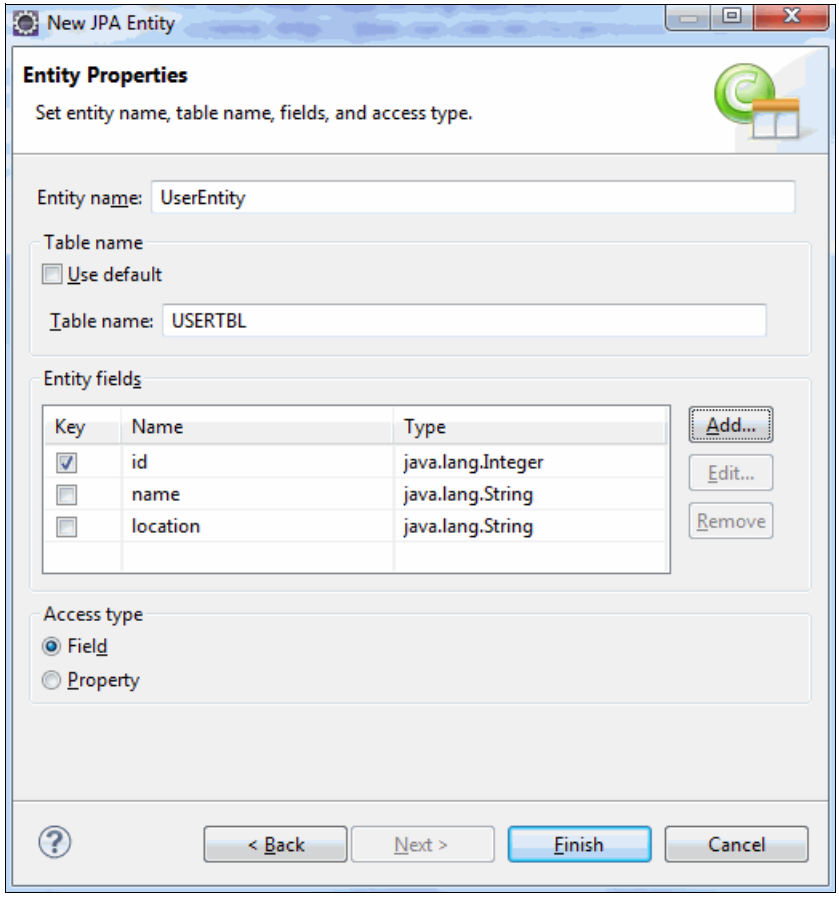


Figure 5-9 JPA Entity Properties

6. To see automated changes, open the Persistence XML editor in either the design or source view. Notice that the persistence class for UserEntity was automatically added to the `persistence.xml` file. Now, open the source for the UserEntity class. Notice that the `id` field has been annotated with the `javax.persistence.Id` annotation to indicate it is the primary key.
7. The Derby database we are using is automatically generating the `id` field. For JPA to handle the generated `id` field correctly, it needs to be annotated with `javax.persistence.GeneratedValue`. The strategy attribute for the annotation should be set to `GeneratedValue.IDENTITY`. Figure 5-10 on page 101 shows how the annotation should look.

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer id;
```

Figure 5-10 The id field for the UserEntity class

8. Open the `persistence.xml` file again in the design editor. Click the **ITSOWeb persistence** unit. In the details panel, enter `jdbc/ITSO` in the JTA data source field.
9. Add a persistence context reference to the deployment descriptor. Open the `web.xml` file in the design editor. Right-click the **Web Application (ITSOWeb)** element and select **Add** → **Persistence Context Reference**. In the details panel, enter `jpasample/entitymanager` in the Persistence Context Reference Name field and `ITSOWeb` in the Persistence Unit Name field.
10. Create a utility class to handle the actual persistence by right-clicking the project and selecting **New** → **Class**. Enter `com.itso.example` for the package name and `Persistor` for the class name. In the source editor, create two methods: retrieveVisitors and persistVisitor.

    The retrieveVisitors method looks up an entity manager using the JNDI name `java:comp/env/jpasample/entitymanager`. It then uses the EntityManager to execute a query that selects all UserEntity objects and returns the results.

    The persistVisitor method accepts two String arguments, username and location. It first looks up a UserTransaction object and invokes the **`begin`** method to start a global transaction. The method then creates a new UserEntity object using the `username` and `location` variables. It then looks up the EntityManager and persists the UserEntity object. Finally, the **`commit`** method is invoked on the UserTransaction to commit the transaction. The completed Persistor class is shown in Figure 5-11 on page 102.

```
public class Persistor {

    private static final String JNDI_NAME = "java:comp/env/jpasample/entitymanager";

    public List<UserEntity> retrieveVisitors() throws NamingException {
        // Look up the EntityManager in JNDI
        Context ctx = new InitialContext();
        EntityManager em = (EntityManager) ctx.lookup(JNDI_NAME);
        // Compose a JPQL query
        String query = "SELECT u FROM UserEntity u";
        Query q = em.createQuery(query);

        // Execute the query
        @SuppressWarnings("unchecked")
        List<UserEntity> users = q.getResultList();

        return users;
    }

    public void persistVisitor(String username, String location)
            throws NamingException, NotSupportedException, SystemException,
            IllegalStateException, SecurityException, HeuristicMixedException,
            HeuristicRollbackException, RollbackException {
        Context ctx = new InitialContext();
        // Before getting an EntityManager, start a global transaction
        UserTransaction tran = (UserTransaction) ctx
                .lookup("java:comp/UserTransaction");
        tran.begin();

        // Now get the EntityManager from JNDI
        EntityManager em = (EntityManager) ctx.lookup(JNDI_NAME);

        UserEntity user = new UserEntity();
        user.setName(username);
        user.setLocation(location);
        em.persist(user);

        // Commit the transaction
        tran.commit();
    }
}
```

Figure 5-11 Example utility class to handle persistence

11. Create a front end for the JPA application. Begin by creating a new JSP file named `jpaInput.jsp`. The JSP contains a form with two input fields and a Submit button. Figure 5-12 on page 103 shows the completed JSP file.

```
<html>
<body>
<form action="JPAServlet" method="POST">
Name:
<input type="text" name="name"/>
<BR>
Location: <input type="text" name="location"/>
<BR>
<input type="submit" value="Submit New Visitor"/>
</form>
</body>
</html>
```

Figure 5-12 jpaInput.jsp file

12. Create a servlet named JPAServlet. The **doGet** method makes use of the **retrieveVisitors** method on the Persistor class to output a table of names and locations. It also uses a RequestDispatcher to include the `jpaInput.jsp` file. Figure 5-13 shows an example **doGet** method.

```
protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    PrintWriter writer = response.getWriter();
    writer.println("<BODY>");
    writer.println("<H2>Hello, Liberty Developer from JPA.</H2>");

    writer.println("These are our previous visitors: ");
    writer.println("<TABLE border=5>");
    writer.println("<TH>Name</TH><TH>Location</TH>");
    try {
        List<UserEntity> users = persistor.retrieveVisitors();

        for (UserEntity user : users) {
            writer.println("<TR><TD>" + user.getName() + "</TD>");
            writer.println("<TD>" + user.getLocation() + "</TD></TR>");
        }
    } catch (NamingException ex) {
        throw new ServletException(ex);
    }
    writer.println("</TABLE>");

    RequestDispatcher rd = request.getRequestDispatcher("jpaInput.jsp");
    rd.include(request, response);

    writer.println("</TABLE>");
    writer.println("</BODY>");

}
```

Figure 5-13 JPAServlet doGet method

The JPAServlet also has a **doPost** method that uses the name and location parameters on the request to persist a new UserEntity object. Figure 5-14 on page 104 shows a sample **doPost** method.

```
protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
    PrintWriter writer = response.getWriter();
    String name = request.getParameter("name");
    String location = request.getParameter("location");
    writer.println("<BODY>");
    try {
        persistor.persistVisitor(name, location);
        writer.println("Added visitor " + name + " from " + location);
    } catch (Exception ex) {
        writer.println("Something went wrong: " + ex.getMessage());
    }

    writer.println("<FORM action=\"JPAServlet\" method=\"GET\">");
    writer.println("<INPUT type=\"submit\" value=\"Home\"/>");
    writer.println("</FORM>");
    writer.println("</BODY>");

}
```

Figure 5-14 JPAServlet doPost method

13. Finally, test the application by visiting:

    `http://localhost:9080/ITSOWeb/JPAServlet`

    You should see output similar to Figure 5-15.

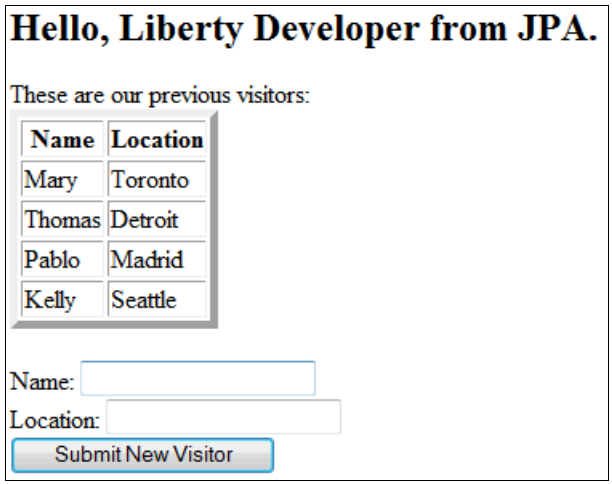


Figure 5-15 Output from the JPA application

5.2.2 JPA applications outside the Liberty profile developer tools

You can develop JPA applications outside of the Liberty profile developer tools. If you do so, then you must enable the `jpa-2.0` feature in the server configuration as described in 1.3.1, “Feature management” on page 12.

The following list notes the key steps for creating a JPA application outside of the Liberty developer tools environment:

- Creating the UserEntity class that maps to the database table `USERTBL`.
- Creating the `persistence.xml` file.
- Adding a persistence context reference to the deployment descriptor.

As with other types of applications, the dynamic nature of the Liberty profile server will speed up application development by allowing you to make iterative updates to your JPA application without having to restart the application or server.

6

Configuring application security

Security is an essential component of any enterprise-level application. In this chapter we provide you with a basic introduction to security using the Liberty profile.

The following topics are covered:

- Enabling SSL
- HTTPS redirect
- Form Login

It is beyond the scope of this book to describe all aspects of security and how to configure them. For further details about security and how to implement specific security models not covered by this chapter (such as LTPA or SSO) refer to the information center at:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/twlp_sec.html

6.1 Enabling SSL

You can configure the Liberty profile server to provide secure communication between a client and the server by enabling SSL. To establish secure communication, a certificate and an SSL configuration must be specified.

The keystore and certificate can be created either using the WebSphere Developer Tools or from the command line. For development you will usually use a self-signed certificate as shown in our examples.

Important: On production servers, import a certificate from a trusted provider to the server keystore rather than using a self-signed certificate.

6.1.1 Configuration using the WebSphere Developer Tools

Use the WebSphere Developer Tools to add the keystore and self-signed certificate to your server using the following steps:

1. Open the Servers view in Eclipse Workbench.
2. Right-click your server and select **Utilities** → **Create SSL Certificate,** as shown in Figure 6-1.

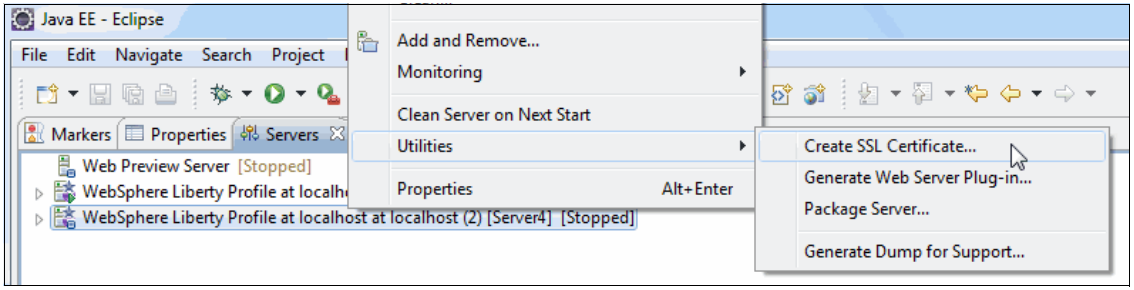


Figure 6-1 Selecting to crete the SSL Certificate using WebSphere Developer Tools

3. You will be prompted to create a password for your keystore and optionally set the validity period or subject for the certificate, as shown in Figure 6-2 on page 109.

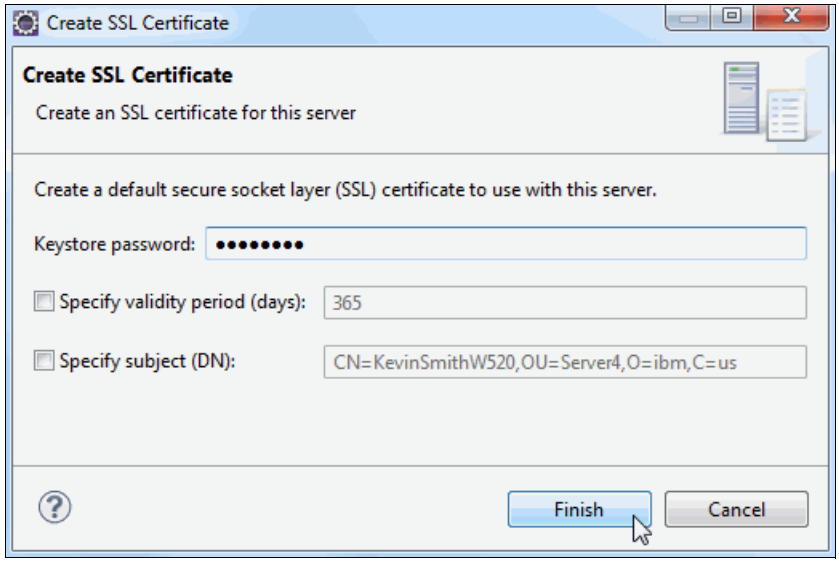


Figure 6-2 Entering values for the keystore and certificate

4. Enter the required values and click **Finish.** The Console window will show the output of the command, as shown in Figure 6-3.

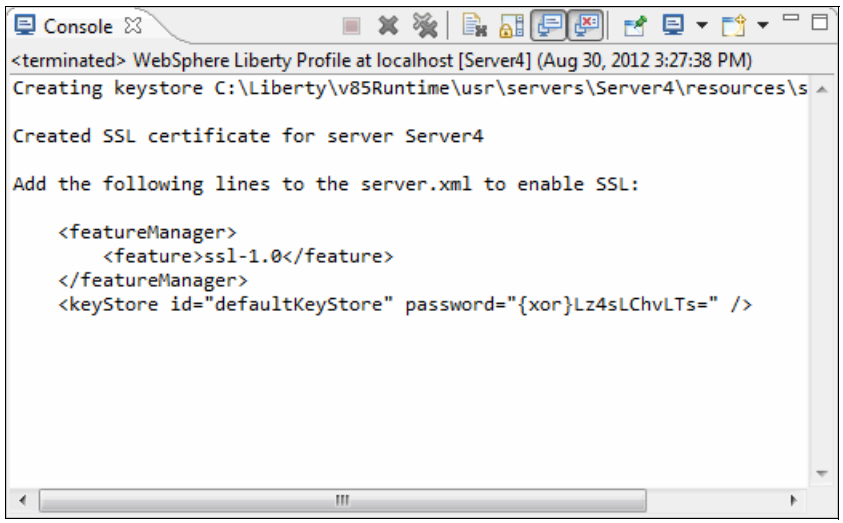


Figure 6-3 Output from the creation of the SSL Certificate

5. The output contains the values that need to be entered into the server configuration. You can either paste the required values directly into the

`server.xml` configuration file (in which case your configuration is complete and you can proceed to “HTTPS redirect”) or alternatively use the Design view to be guided through the configuration as shown in the following steps:

6. In the Servers view, expand the server until you see the Feature Manager entry. Double-click the **Feature Manager** or right-click and select **Open.**
7. The Feature Manager panel will be displayed. Under Feature Manager Details click **Add**, as shown in Figure 6-4.

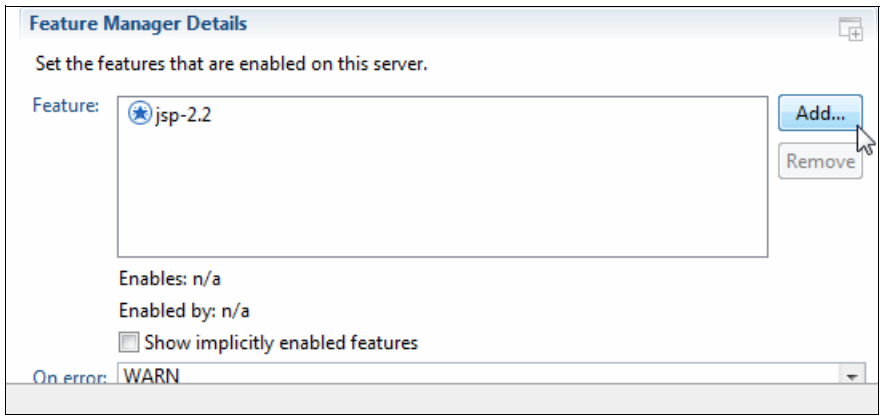


Figure 6-4 Adding a new feature using the Feature Manager

8. In the Add Feature Dialogue, select **ssl-1.0** from the list and click **OK.**
9. On the left side of the Feature Manager panel you will see the Configuration Structure being displayed. To add in the keystore, select **Server Configuration** and click **Add**, as shown in Figure 6-5.

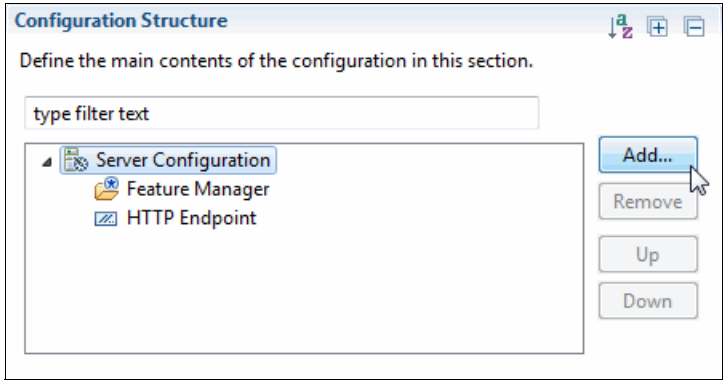


Figure 6-5 Adding new items to the configuration structure

10. Scroll down the list until you find the keystore entry. Select **keystore** and then click **OK**.
11. The keystore configuration will now appear under the Server Configuration. The keystore details are shown in the right side of the panel (if they are not shown, select **Keystore** in the Server Configuration).
12. In the Id field enter `defaultKeyStore`.
13. Click **Edit** next to the Password field and enter the password you used when creating your keystore during step 3 on page 108.
14. Your configuration is now complete. Save the changes to the server configuration. The server (if started) will now detect the new settings and the SSL port will be opened.
15. If you have applications installed on your server, you will now be able to access them by specifying `https://` and pointing your browser to the https port defined in your server configuration, HTTP Endpoint.

6.1.2 Configuration using the command line

To create the keystore and generate the self-signed certificate using the command line, you need to use the **securityUtility** command located in `${wlp.install.dir}/bin`. Example 6-1 shows the command and the expected output.

Example 6-1 Using the securityUtility command

```
C:\Liberty\v85Runtime\bin>securityUtility createSSLCertificate
--server=myThirdServer --password=passwOrd --validity=365
--subject="CN=liberty,O=IBM,C=US"

Creating keystore
C:\Liberty\v85Runtime\usr\servers\myThirdServer\resources\security\key.
jks

Created SSL certificate for server myThirdServer

Add the following lines to the server.xml to enable SSL:

    <featureManager>
        <feature>ssl-1.0</feature>
    </featureManager>
    <keyStore id="defaultKeyStore" password="{xor}Lz4sLChvLTs=" />
```

The output from the command contains the information to be added into your server configuration, to enable SSL. Cut and paste this output into your `server.xml` file. The server will automatically detect the changes and activate the SSL port.

6.2 HTTPS redirect

With SSL enabled the client will, in most cases, still be able to access the application on both the secure and nonsecure port. HTTPS redirect allows you to always direct the client to the secure HTTPS port even if the client has requested the standard nonsecure port.

This section assumes you are starting with the web application created in Chapter 3, “Developing and deploying web applications” on page 43. If you do not have this application in your project workspace you can find a completed version included in the download material for this book. For more information see Appendix A, “Additional material” on page 147.

To access and edit the available web applications to work with HTTPS redirect, complete the following steps:

1. In Project Explorer, expand your web application and right-click the deployment descriptor. Select **Open With** → **Web Application 3.0 Deployment Descriptor Editor**. If you do not see Web Application 3.0 Deployment Descriptor Editor in the menu, select **Other** and select the correct editor.
2. To enable HTTPS redirect, add a security constraint to your application. To do this, select **Web Application** in the Overview section and then click **Add**, as shown in Figure 6-6 on page 113.

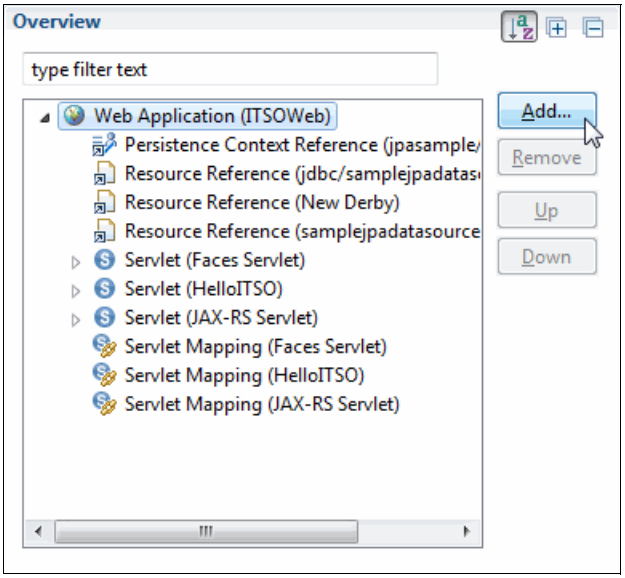


Figure 6-6 Adding a security constraint to the deployment descriptor

3. In the Add Item window, scroll down and select **Security Constraint**, then click **OK**.
4. The Security Constraint will now appear in the Overview section. Next, define the Web Resource Collection for the Security Constraint. This identifies which addresses are covered by the Security Constraint. Expand the Security Constraint in the Overview section and select **Web Resource Collection**.
5. The details panel, to the right of the Editor, allows you to enter the details of the Web Resources. Give the Web Resource Collection a name and add in the URL Patterns to be used, as shown in Figure 6-7 on page 114.

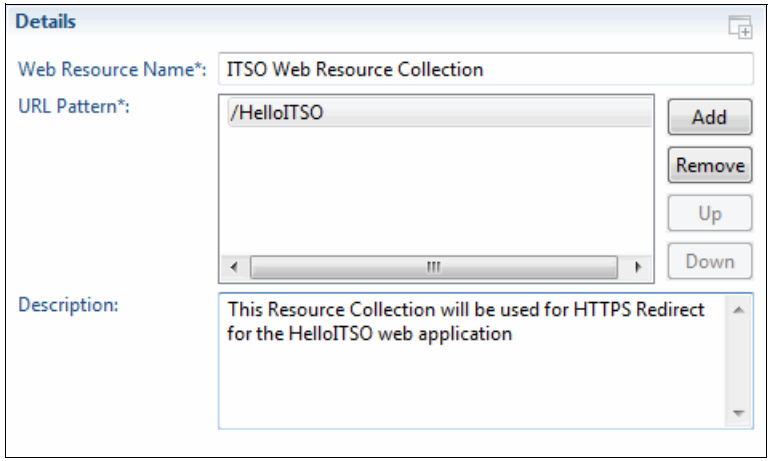


Figure 6-7 Entering details for the Web Resource Collection

6. Next specify the details of the Security Constraint. In the Overview section select **Security Constraint**. The details panel will now allow you to define the Security Constraint.
7. Give the Security Constraint a name. Under the User Data Constraint section select **CONFIDENTIAL** for the transport guarantee, as shown in Figure 6-8 on page 115. This specifies that we will only allow secure communication with the web resource collection.

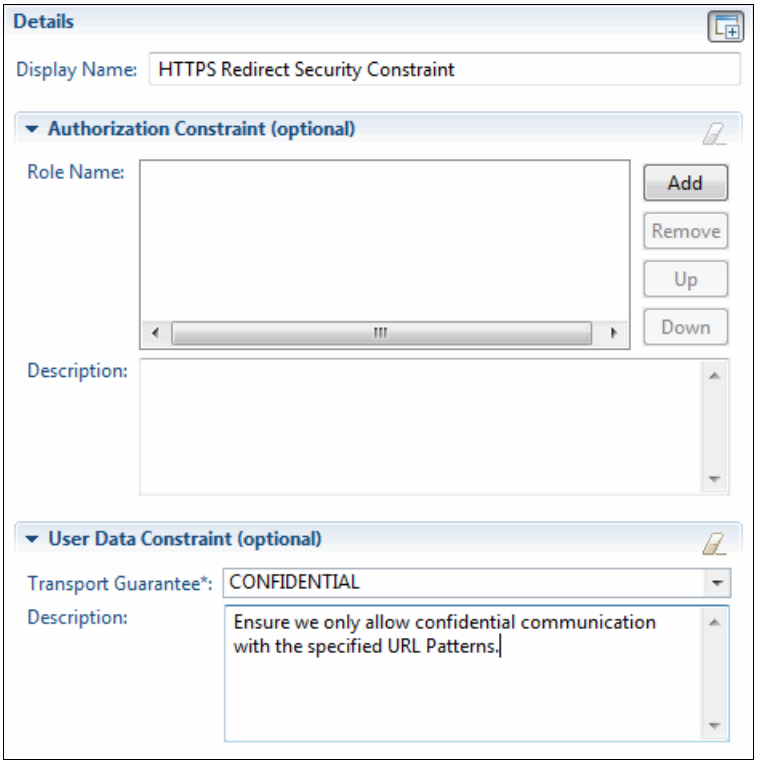


Figure 6-8 Specifying the confidential transport

8. Our example currently applies the HTTPS redirect to all communications with the addresses defined in the Web Resource Collection. To only apply this to specific HTTP methods, right-click the **Web Resource Collection** in the Overview section and select **Add** → **HTTP Method**. We restrict our HTTPS redirect to GET requests only by entering `GET` for the HTTP Method.
9. HTTPS redirect is now fully configured. Save your changes and WebSphere Developer Tool will automatically update the server with the new version of your application. If you now attempt to access the application at:

   `http://localhost:9080/ITSOWeb/HelloITSO`

   you will notice you are always redirected to:

   `https://localhost:9443/ITSOWeb/HelloITSO`

6.3 Form Login

One of the most common ways to provide an authentication method to users of a web application is to use form-based login. This presents the users with a form where they can provide their login credentials. This section describes how to update your application and server to provide the form-based login.

This section expands on the HTTPS redirect example in 6.2, “HTTPS redirect” on page 112.

6.3.1 Defining the Basic User Registry

The first step in configuring form login is to create and define the actual registry that will be used for authentication. In our example, we use a Basic User Registry. However, the Liberty profile does support other registries, including Active Directory and LDAP. The following website provides additional information on configuring the Liberty profile to work with these other registries:

`http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/twlp_sec_authenticating.html`

Use the following steps to create and define the registry for authentication using a Basic User Registry:

1. Update the `server.xml` configuration file with the form login configuration. Open the server view in your Eclipse workbench and double-click the Server Configuration for your server. This will open your `server.xml` file. Switch to the design tab if it is not already selected.
2. In the Configuration Structure section on the left, select the **Feature Manager**. The Feature Manager details will be displayed on the right, as shown in Figure 6-9 on page 117.

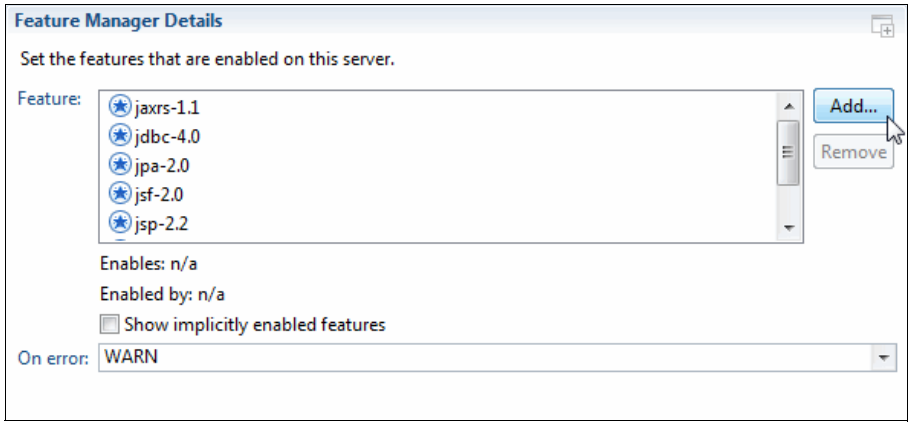


Figure 6-9 Viewing the Feature Manager details

3. If the appSecurity-1.0 feature is not already in the list of features, click **Add** and select the **appSecurity-1.0** feature.
4. We will add the basic registry that will be used for our authentication. Under the Configuration Structure, select **Server Configuration** and click **Add**, as shown in Figure 6-10.

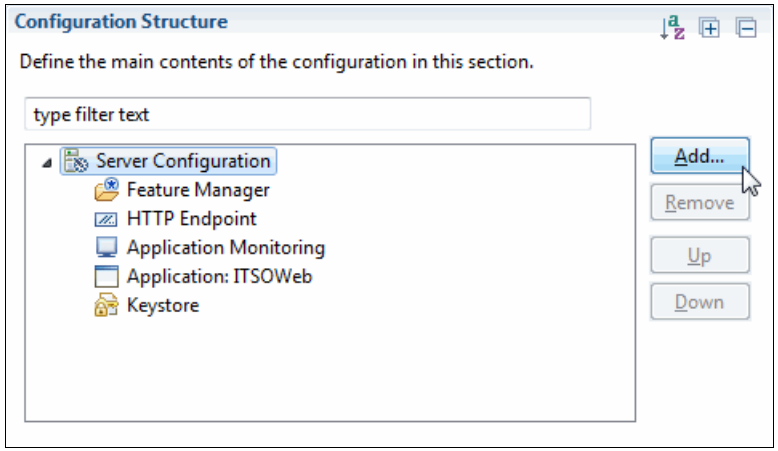


Figure 6-10 Adding items to the server configuration

5. Select **Basic User Registry** and click **OK.**
6. Select **Basic User Registry** in the Configuration Structure. Under the Basic User Registry Details section on the right, enter an ID and Realm name for your registry, as shown in Figure 6-11 on page 118.

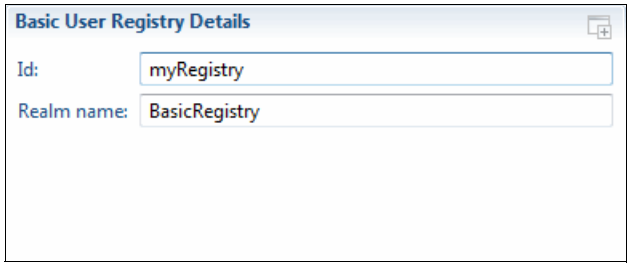


Figure 6-11 Entering the Basic Registry details

7. Now that we have our basic user registry, we need to add users and groups to it. Click **Add**.
8. In the resulting window, select **User** and click **OK.**
9. A user object will now appear under your basic user registry. Click the user object. On the right, under User Details, enter a username of `User1` and then click **Edit** to enter a password, as shown in Figure 6-12.

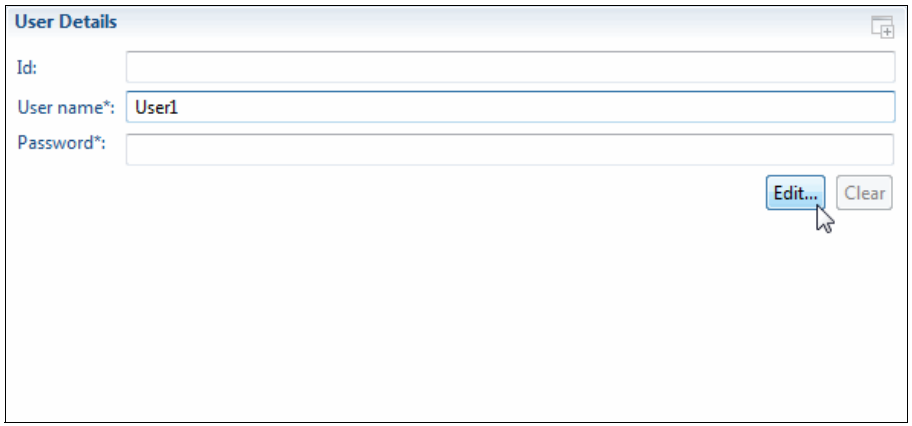


Figure 6-12 Adding a new User

10. Enter the password and password confirmation and click **OK**. You have now created a user for the basic registry. Repeat the process and add another user called `User2`**.**
11. Then add a security group to the basic user registry. Select the **Basic User Registry** and click the **Add...** button.
12. Select **group** from the Add Item window and click **OK.**
13. A group element will appear under your basic user registry. Ensure it is selected and under Group Details enter `group1` as the group name.

14. We now need to add users to the group. Select the group under the Configuration Structure and click **Add**. In the Add Item window select **member** and click **OK.**
15. Select the member element that has appeared under the group. In the Group Member Details panel, enter `User1` for the User name.
16. Save the changes. Your configuration Structure should now contain the basic user registry structure, as shown in Figure 6-13.

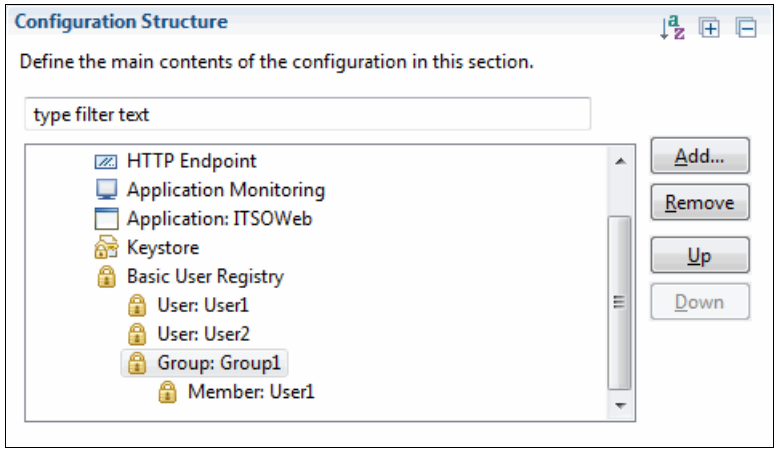


Figure 6-13 The completed basic user registry

6.3.2 Updating an application to support Form Login

To use Form Login for your application, you will need to make a number of changes to the application and its configuration. These changes are needed to allow the user to authenticate. The following sections will describe the changes in detail.

Adding the login page

The user of the application will need to provide their login credentials to be authenticated. To allow them to do this, provide a login page. This login page needs to contain a form which requests the username and password and must always have the *`j_security_check`* action.

Example 6-2 shows how to code the form into the HTML page.

Example 6-2 The login form

```
<form method="POST" action="j_security_check">
<strong> Enter user ID and password: </strong>
```

```
<BR>
<strong> User ID</strong> <input type="text" size="20"
name="j_username">
<strong> Password </strong>  <input type="password" size="20"
name="j_password">
<BR>
<strong> Click submit to authentice: </strong>
<input type="submit" name="login" value="Login">
</form>
```

Use the `j_username` input field to obtain the user name and use the `j_password` input field to obtain the user password.

When a user requests a webpage that requires authentication, the web server stores the original request and displays the login form. When the login form is completed and the user has been successfully authenticated, the web server redirects the call to the original request.

Adding a login error page

If the authentication of a user fails, you need to be able to advise the user of the failure. This is done by adding a login error page to your application. Example 6-3 shows an example of an error page in a JSP file.

Example 6-3 An example login error page

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
<head><title>A Form login authentication failure
occurred</head></title>
<body>
<H1><B>A Form login authentication failure occurred</H1></B>
<P>Authentication might fail for one of many reasons. Some
possibilities include:
<OL>
<LI>The user ID or password might have been entered incorrectly; either
misspelled or the wrong case was used.
<LI>The user ID or password does not exist, has expired, or has been
disabled.
</OL>
</body>
</html>
```

Adding a form logout page

Form logout is a mechanism to log out without having to close all web browser sessions. After logging out, access to a protected web resource will require re-authentication.

The form logout page is an HTML or JSP file that you include with your Web application. This logout page contains a form with a special post action. Example 6-4 provides a sample logout page.

Example 6-4 A Sample logout page

```
<!DOCTYPE HTML PUBliC "-//W3C/DTD HTML 4.0 Transitional//EN">
<html>
<META HTTP-EQUIV = "Pragma" CONTENT="no-cache">
<title>Logout Page </title>
<body>
<h2>Sample Form Logout</h2>
<form method="POST" action="ibm_security_logout" name="logout">
<BR>
<strong> Click this button to log out: </strong>
<input type="submit" name="logout" value="Logout">
<input type="HIDDEN" name="logoutExitPage" value="/login.html">
</form>
</body>
</html>
```

The logoutExitPage specifies where the user is to be redirected after logging out.

Configuring the application

The final changes needed to the web application are in the configuration. The web application needs to define the login mechanism and also restrict access to the relevant endpoints.

Although you do not need to do the HTTPS redirect as part of the form login, it is advised because it ensures confidentiality of the login credentials. The following steps will expand upon the configuration you set up in 6.2, “HTTPS redirect” on page 112 to define the login mechanism and limit the access to the relevant endpoints:

1. In Project Explorer in your Eclipse Workbench, expand your web application and right-click the deployment descriptor. Select **Open With** → **Web Application 3.0 Deployment Descriptor Editor**.
2. The first configuration that needs to be added is a login configuration, which defines the login mechanism. In the Overview section of the Deployment

Descriptor Editor, select **Web Application** and click **Add**, as shown in Figure 6-14.

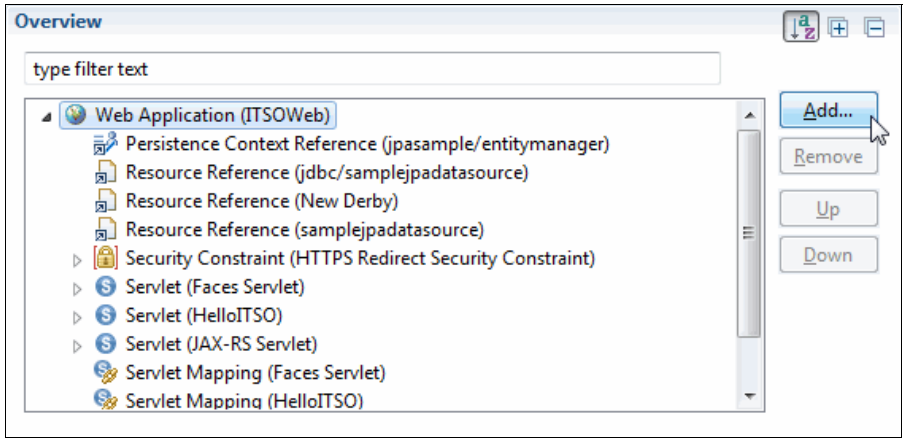


Figure 6-14 Adding a login configuration

3. Select **Login Configuration** from the list and click **OK**.
4. Click the login configuration. You can now define how the user will log in by updating the properties for the login configuration, as shown in Figure 6-15.

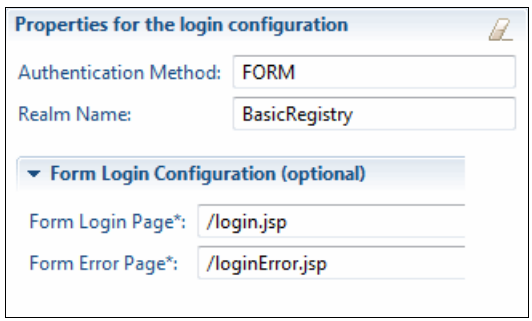


Figure 6-15 Defining the login configuration settings

Note: The realm name is an optional attribute that is used for differentiating between different repositories. If you have multiple repositories in your server, each one can have a realm name. You can then use the realm name to identify the repository to use for authentication. If you have only one repository you can leave this value blank.

5. You now need to restrict your endpoint to only allow certain users to access the resource. This is done by adding a security constraint and adding an authorization constraint to it. We already have a security constraint that we added in 6.2, “HTTPS redirect” on page 112. We will use this and add the authorization constraint to it. Select the **Security Constraint** from within the Overview section, as shown in Figure 6-16.

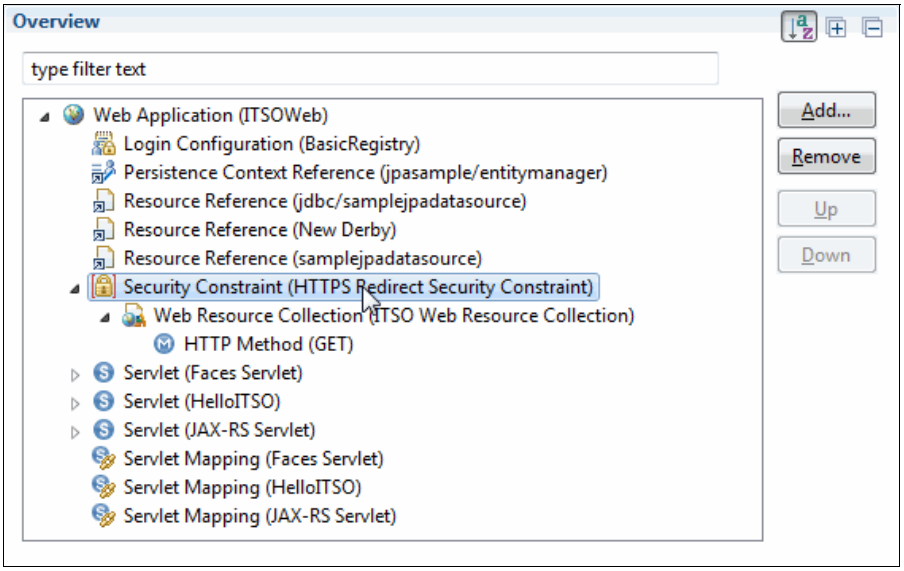


Figure 6-16 Selecting the Security Constraint

6. In the detail section, under Authorization Constraint, click **Add** to the right of the Role Name. This adds a new role that is required for access to the resource. Give the role a name, as shown in Figure 6-17 on page 124.

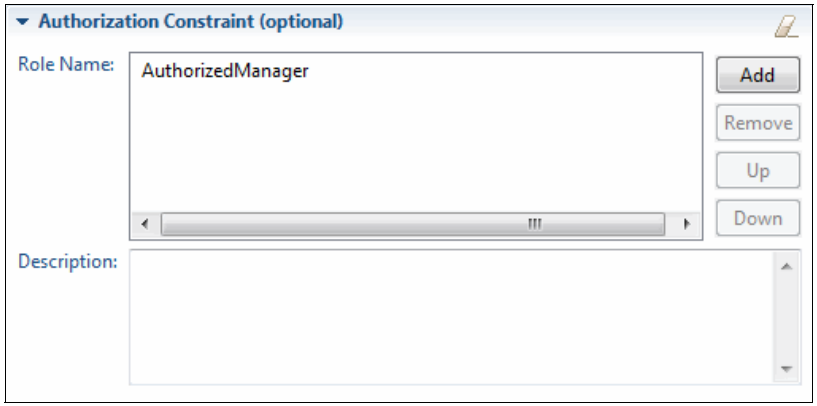


Figure 6-17 Adding a new authorization role

Note: The Role Name within your application is a logical name and can be any value you like. In “Defining bindings between the application and the server” you will add mappings between your application roles and the users and groups defined in your user registry.

7. The application changes have now been completed. Save all changes.

6.3.3 Defining bindings between the application and the server

The final step in enabling Form Login is to map the roles defined within the application to the groups and users in your user registry. To do this you need to add an application binding in your server configuration using the following steps:

1. Open the Server view in Eclipse and double-click **Server Configuration** for your server. This opens your `server.xml` file. Switch to the design tab if it is not already selected.
2. In the Configuration Structure, select your application and click **Add,** as shown in Figure 6-18 on page 125.

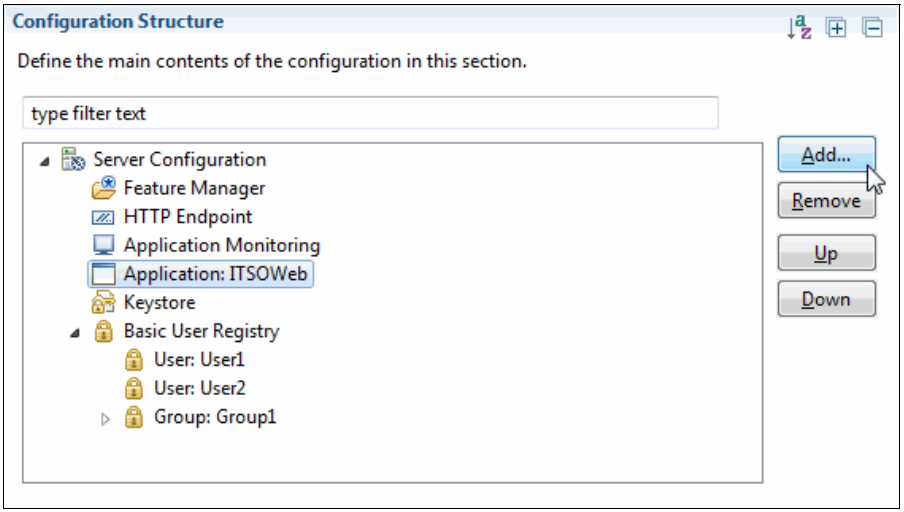


Figure 6-18 Adding an application binding

3. Select **Application Binding** from the list and click **OK**.
4. We now need to specify the role to map. This should be the role that you defined in your application. Select the Application Binding under the Configuration Structure and click **Add**.
5. Select **security-role** from the list and click **OK**.
6. Make sure that the security role is selected in the Configuration Structure and fill in the security role name, as shown in Figure 6-19.

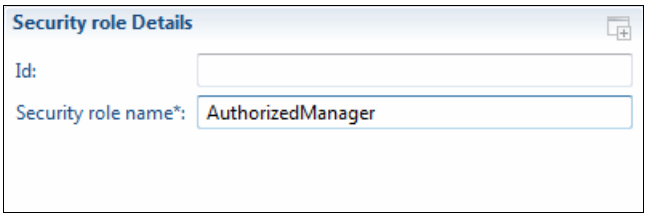


Figure 6-19 Defining the security role name

7. Map the security role with the users and groups that are to be considered part of this role. In our example we are going to map `group1` to the `AuthorizedManager` role. Select the security-role under the Configuration Structure and click **Add**.
8. Select **group** from the list and click **OK**.

Note: As well as user and group you can also select the special-subject option to assign access to either everyone or all authenticated users.

9. Select the **Group element** in the Configuration Structure and under Group Details enter the name of the group that is to be allowed access to the resource, as shown in Figure 6-20.

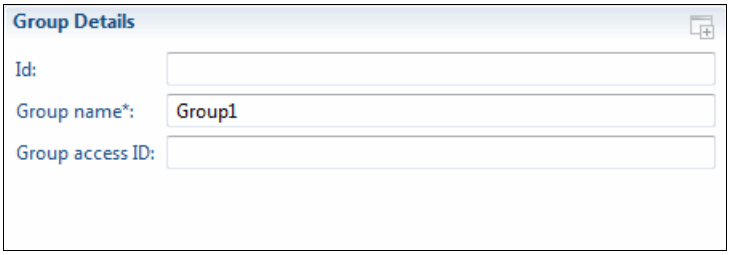


Figure 6-20 Entering the group details

10. Your Forms Login configuration is now complete. Save all changes.

If you now access your application, you will be taken to the login window. When you enter the user credentials, only `User1`, which is in `Group1`, will have access to the application.

7

Serviceability and troubleshooting

The Liberty profile has several tools to help identify problems with the server and the applications deployed to it. In this chapter we cover the following tools:

- Trace
- Server dump
- MBeans and JConsole

7.1 Trace

The Liberty profile has the ability to produce a variety of traces that enable you to debug issues with the server and your applications. By default, limited information is reported, however; you can request additional trace information if required. In this section we describe where to find the output and how to configure what trace data is collected.

7.1.1 Inspecting the output logs

The Liberty profile server records limited information by default. This basic information is useful for debugging common configuration issues. You can view the output logs by opening the *${server.config.dir}*/logs/messages.log file. Example 7-1 shows a sample output from a server starting.

Example 7-1 Sample content of the message.log file

```
[05/09/12 16:48:15:350 EDT] 00000001
.ibm.ws.kernel.launch.internal.platform.FrameworkManagerImpl A CWWKE0001I: The
server myFirstServer has been launched.
[05/09/12 16:48:17:063 EDT] 00000001
.ibm.ws.kernel.launch.internal.platform.FrameworkManagerImpl I CWWKE0002I: The
kernel started after 1.866
[05/09/12 16:48:17:063 EDT] 00000028
com.ibm.ws.kernel.feature.internal.FeatureManager            I CWWKF0007I:
Feature update started.
[05/09/12 16:48:17:309 EDT] 0000001d
com.ibm.ws.security.internal.SecurityReadyService            I CWWKS0007I: The
security service is starting...
[05/09/12 16:48:17:449 EDT] 0000001d com.ibm.ws.tcpchannel.internal.TCPChannel
I CWWKO0219I: TCP Channel defaultHttpEndpoint has been started and is now
listening for requests on host 127.0.0.1  (IPv4: 127.0.0.1) port 9080.
[05/09/12 16:48:18:162 EDT] 00000035 com.ibm.ws.tcpchannel.internal.TCPChannel
I CWWKO0219I: TCP Channel defaultHttpEndpoint-ssl has been started and is now
listening for requests on host 127.0.0.1  (IPv4: 127.0.0.1) port 9443.
[05/09/12 16:48:18:411 EDT] 0000001d
com.ibm.ws.app.manager.internal.monitor.DropinMonitor        A CWWKZ0058I:
Monitoring dropins for applications.
[05/09/12 16:48:18:456 EDT] 00000021
com.ibm.ws.app.manager.internal.statemachine.StartAction     I CWWKZ0018I:
Starting application ITSOWeb.
[05/09/12 16:48:18:560 EDT] 00000033
com.ibm.ws.security.internal.SecurityReadyService            I CWWKS0008I: The
security service is ready.
```

```
[05/09/12 16:48:18:562 EDT] 00000033
com.ibm.ws.security.token.ltpa.internal.LTPAKeyCreator      I CWWKS4105I: LTPA
configuration is ready after 0.25 seconds.
[05/09/12 16:48:19:765 EDT] 00000021
com.ibm.ws.webcontainer.osgi.webapp.WebGroup                 I SRVE0169I:
Loading Web Module: ITSOWeb.
[05/09/12 16:48:19:766 EDT] 00000021 com.ibm.ws.webcontainer
I SRVE0250I: Web Module ITSOWeb has been bound to default_host.
[05/09/12 16:48:19:767 EDT] 00000021 com.ibm.ws.http.internal.VirtualHostImpl
A CWWKT0016I: Web application available (default_host):
http://localhost:9080/ITSOWeb/*
[05/09/12 16:48:19:768 EDT] 00000021 com.ibm.ws.webcontainer.osgi.WebContainer
I SRVE9998I: Application ITSOWeb added to web container.
[05/09/12 16:48:19:769 EDT] 00000021
com.ibm.ws.app.manager.internal.statemachine.StartAction     A CWWKZ0001I:
Application ITSOWeb started in 1.313 seconds.
[05/09/12 16:48:19:835 EDT] 00000028
com.ibm.ws.kernel.feature.internal.FeatureManager            I CWWKF0008I:
Feature update completed in 2.772 seconds.
[05/09/12 16:48:19:835 EDT] 00000028
com.ibm.ws.kernel.feature.internal.FeatureManager            A CWWKF0011I: The
server myFirstServer is ready to run a smarter planet.
```

The log shows the server startup procedure. First, the kernel starts. Then the feature manager initializes and reads the configuration from the `server.xml` file. The server is configured to listen on a given port. Then the ITSOWeb application is started and finally the server is ready to serve the application.

7.1.2 Configuration of additional trace

A typical task when working with applications is to enable and configure the logging properties of the runtime server to inspect for problems. The Liberty profile server can be configured to gather debug information for both runtime issues and application code.

The Liberty profile server, when received, is configured to capture a minimal amount of trace information. You can modify this default by specifying properties in the server configuration file or `bootstrap.properties` file. OSGi logging output is intercepted and delivered through the trace support. There is also interception of `java.util.logging` output.

To learn more about configuring the logging service, refer to the information center at the following website:

http://publib.boulder.ibm.com/infocenter/radhelp/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/rwlp_logging.html

Configuring trace in the servers configuration

To configure additional trace in the servers configuration, follow these simple steps:

1. Open the server view in your Eclipse workbench and double-click **Server Configuration** for your server. This opens your `server.xml` file. Select the **Design** tab.
2. Under the Configuration Structure, select **Server Configuration** and click **Add**, as shown in Figure 7-1.

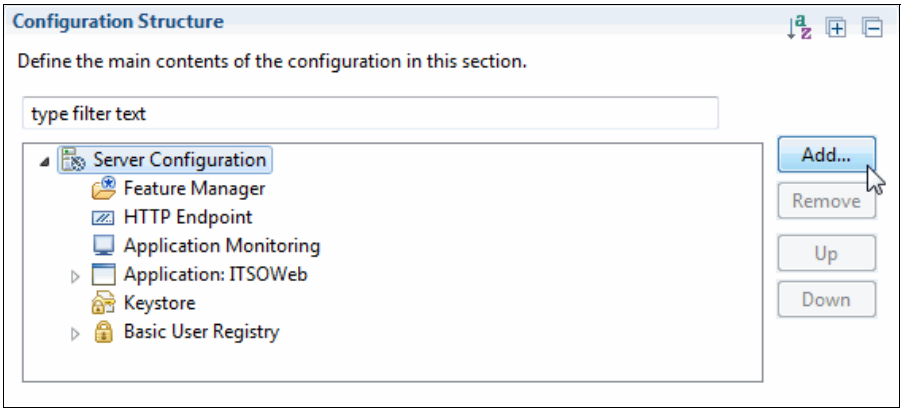


Figure 7-1 Adding the logging option to the server configuration

3. Select **Logging** and click **OK**.
4. Ensure that logging is selected under the Configuration Structure. You can now configure logging in the Logging Details section, as shown in Figure 7-2 on page 131.

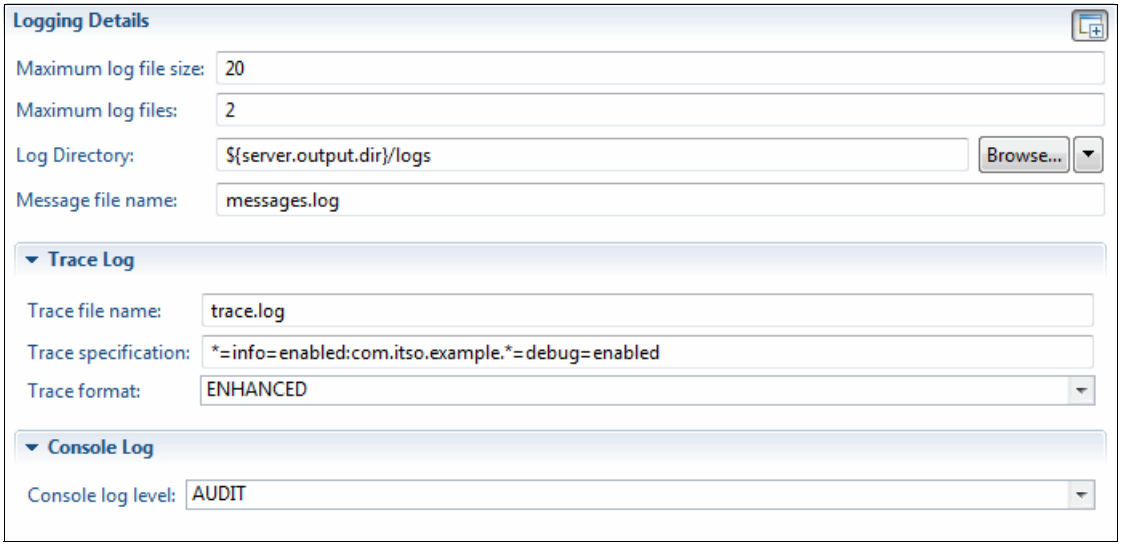


Figure 7-2 Entering the trace details

The Logging Details panel allows you to configure many aspects of logging, including the location of the logs and where they are stored. The Trace Specification defines what information to capture in the logs. In our example, we capture information level messages for all classes. In addition, we chose to capture all debug or higher level messages for any `com.itso.example.*` classes.

For information about how to add logging to your applications, refer to the following WebSphere Application Server Information Center website:

`http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/ttrb_javalogapps.html`

Configuring trace using bootstrap.properties

Using the server configuration file to enable logging is an easy and efficient method. However, the following list notes some cases where you might want to put logging properties into the `bootstrap.properties` file:

- When logging is required before the configuration processing has occurred at server start (this might be required for IBM support for debugging a server start problem).
- When the `server.xml` configuration is shared by multiple server instances, and logging is only required on one server (then the `bootstrap.properties` file for that server can be modified).

Example 7-2 shows a sample `bootstrap.properties` file that enables tracing of any `com.itso.example.*` classes.

Example 7-2 A sample bootstrap.properties file

```
# New trace messages are written to the end of the file (true or false)
com.ibm.ws.logging.trace.append = "true"
# maximum size of e;ach trace file (in MB)
com.ibm.ws.logging.trace.max.file.size = 2
# maximum number of trace files
com.ibm.ws.logging.trace.max.files = 2
# Trace format (basic or advanced)
com.ibm.ws.logging.trace.format = "basic"
# Trace settings string - this string enables all trace
com.ibm.ws.logging.trace.specification =
com.itso.example.*=debug=enabled
```

7.2 Server dump

To capture state information of a Liberty profile server, you can use the **dump** command. This can be useful for problem diagnosis of a Liberty profile server.

The result file generated by the **dump** command contains server configuration, log information, and details of the deployed applications in the `workarea` directory. The **dump** command can be used against a running or stopped server. If the server is running, the following additional information will be gathered:

- State of each OSGi bundle in the server
- Wiring information for each OSGi bundle in the server
- Component list managed by the Service Component Runtime (SCR)
- Detailed information of each component from SCR

The following syntax is an example of running the **dump** command:

```
server dump server1 --archive=c:\tmp\server1_dump.zip
```

You can also create a dump of the Liberty profile server using the WebSphere Developer Tools. This is done by right-clicking the server in the Servers view and selecting **Utilities** → **Generate Dump for Support**.

7.3 MBeans and JConsole

The Liberty profile does not come with any dedicated tools to monitor its runtime, such as the Tivoli® Performance Viewer available in the traditional WebSphere Application Server console. To monitor the Liberty runtime characteristics, administrators and developers can use standard tools for Java runtimes such as the Jconsole.

The Liberty profile offers the `monitor-1.0` feature. This feature allows users to track information about the Liberty profile server runtime, such as JVM, web applications, and the thread pool. To enable this monitoring, add the `monitor-1.0` feature to the `server.xml` configuration so the following MXBeans are available:

- WebSphere:type=JvmStats
- WebSphere:type=ServletStats,name=*
- WebSphere:type=ThreadPoolStats,name=Default Executor

Refer to the following information center website for more information about monitoring the Liberty profile using MXBeans:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.zseries.doc/topics/twlp_mon.html

Jconsole is shipped as part of the Java Runtime package. To run Jconsole against the Liberty profile, start it from the command line, as shown in the following syntax:

```
C:\IBM\WebSphere\AppServer\java\bin>jconsole.exe
```

When Jconsole launches, a welcome page appears. You have to connect to the Java process that runs the Liberty profile. The connection is made by logging in. The Jconsole process must run under the same operating system user ID and use the same Java runtime as the server. As illustrated in Figure 7-3 on page 134, we chose to use the local server for our example.

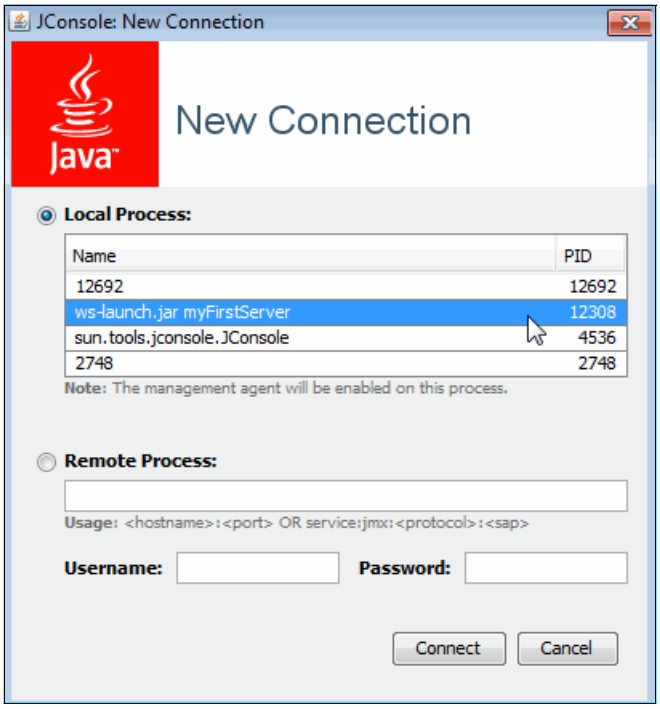


Figure 7-3 Establishing a connection using JConsole

The JConsole will load all runtime information into the tool. This includes the characteristic of the Java runtime, heap size, or processor usage, as illustrated in Figure 7-4 on page 135.

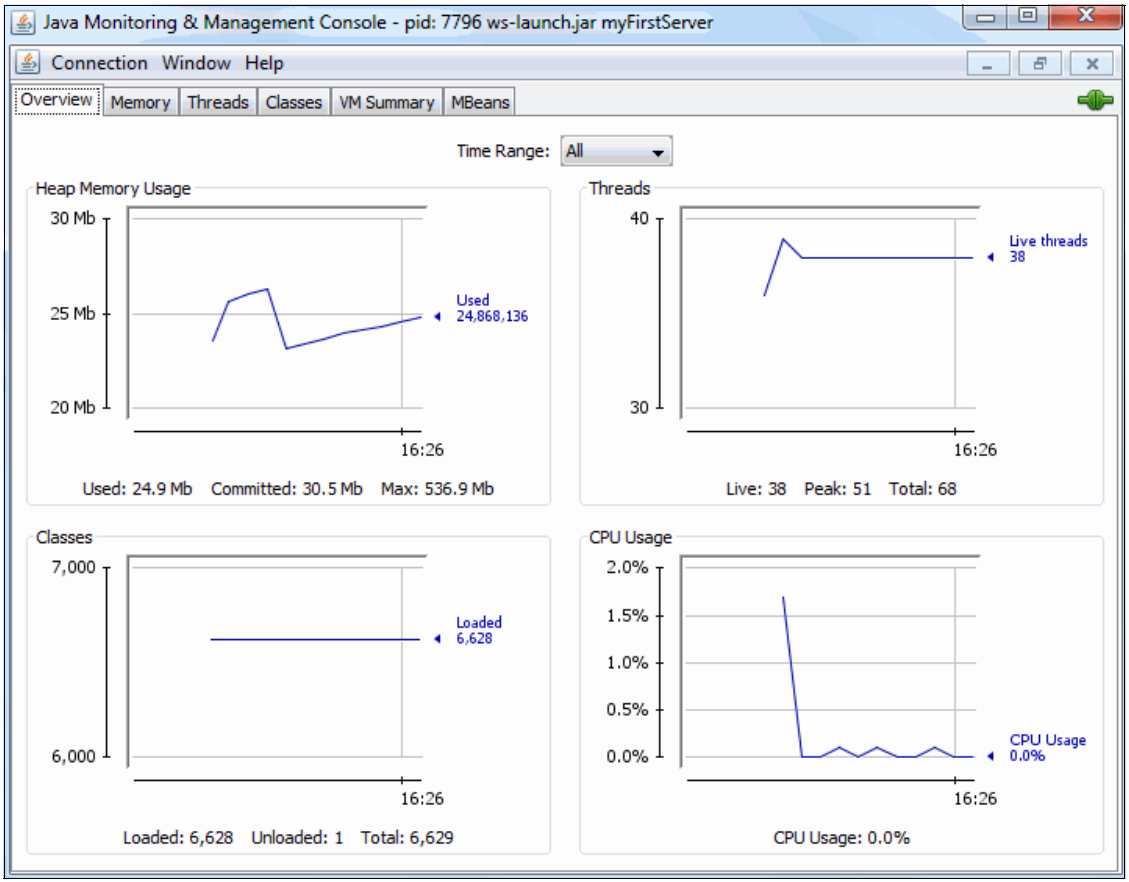


Figure 7-4 Example monitoring output using JConsole

Using the JConsole tool you can also trigger MBeans operations that are part of the Liberty profile. MBeans are available under the MBeans tab. Figure 7-5 on page 136 shows an example of issuing a **`restart`** command on the ITSOWeb application.

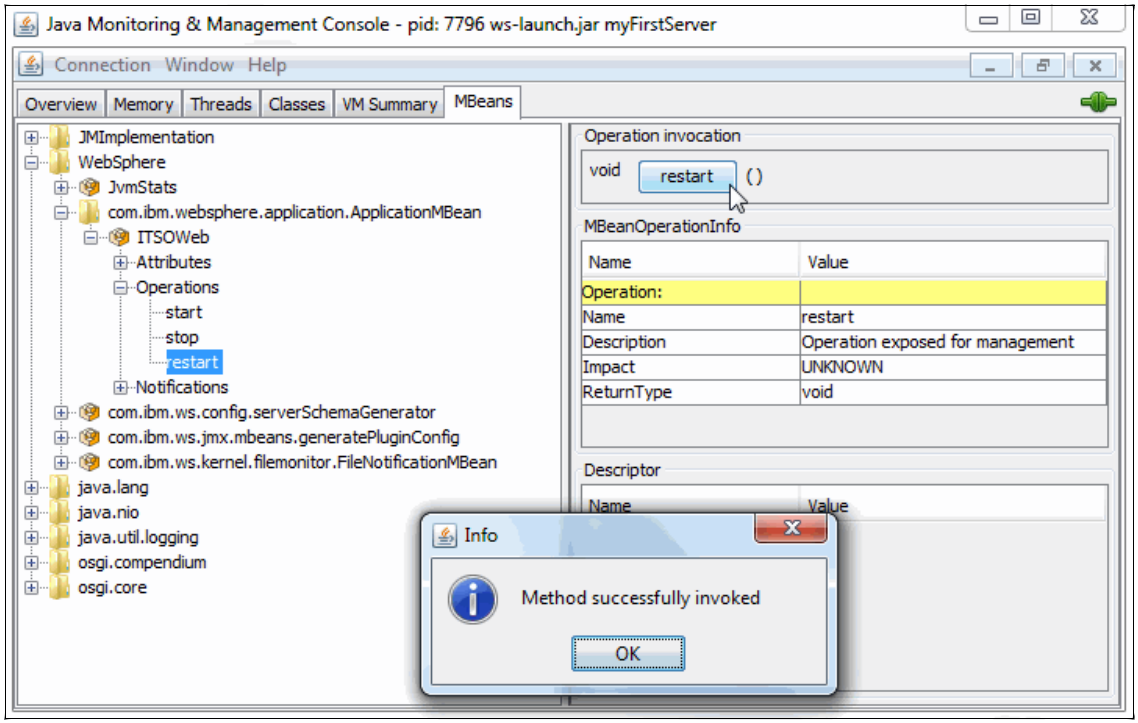


Figure 7-5 Issuing a restart request for an application using MBeans

If you enabled the monitoring infrastructure in Liberty, you can also access additional MXBeans such as **`JvmStats`**, as illustrated in Figure 7-6 on page 137.

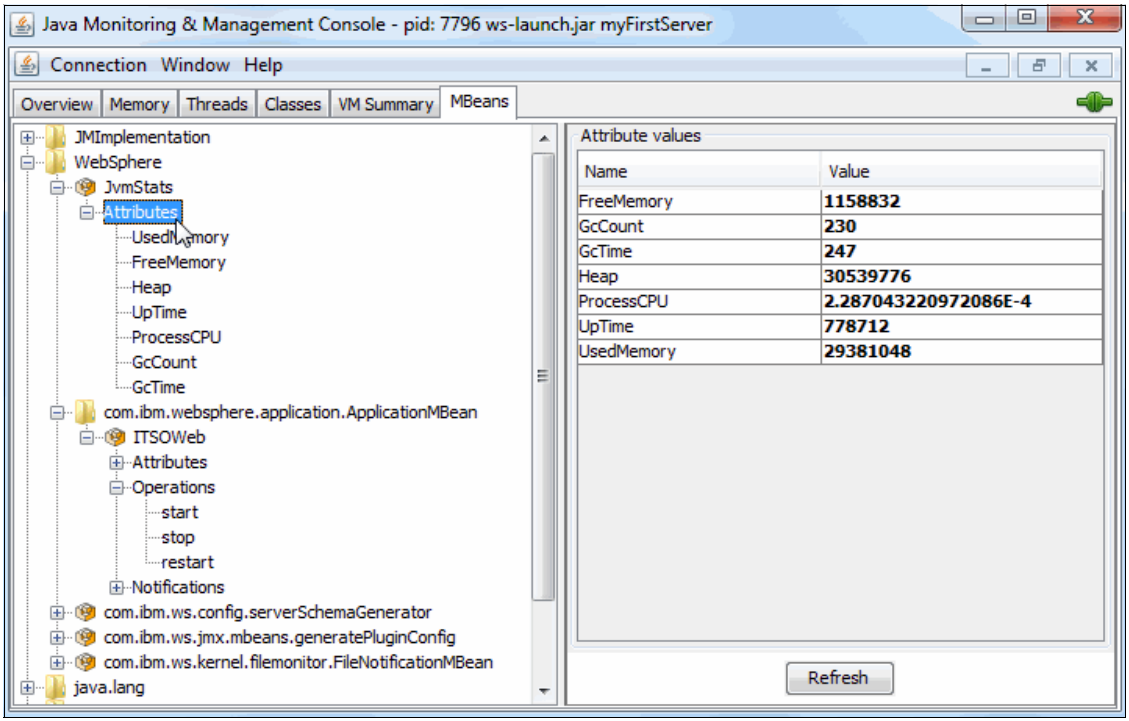


Figure 7-6 Viewing JVMStats using Jconsole

8

From development to production

The Liberty profile server is optimized for development use, but is supported for use in both development and production environments. This chapter describes several considerations for moving the Liberty profile server from a development environment to a production environment.

The chapter contains the following sections:

- Configuring a server for production use
- Using the package utility
- Moving an application to the full profile
- Using the Liberty profile on z/OS

8.1 Configuring a server for production use

The following advanced configuration settings should be considered when using a Liberty profile server in a production environment.

8.1.1 Turn off application monitoring

By default, application directories are monitored for any updates. Because applications are unlikely to change frequently in a production environment and because this monitoring can be resource intensive, the best practice is to disable application monitoring.

Application monitoring can be disabled in the Liberty profile developer tools by editing the server configuration. In the design view, click **Application Monitoring**. In the Application Monitoring Details panel, uncheck the box next to Monitor application drop-in directory. Additionally, you might want to change the value of the drop-down for Application update trigger to `disabled`.

To disable application monitoring, by editing the `server.xml` file, add the following element:

```
<applicationMonitor updateTrigger="disabled" dropinsEnabled="false"/>
```

You can also change the value of the updateTrigger attribute to `mbean`. This allows you to update the application manually using JMX.

8.1.2 Generate web server plug-in configuration

To use the Liberty profile server with an external web server, you need to generate the web server plug-in configuration, whose properties can be specified in the server configuration.

To add plug-in configuration properties in the Liberty profile developer tools, load the `server.xml` file in the Design editor and click the server configuration. Then click **Add** and select **Generate Plug-in**. You can specify properties such as ports in the Generate Plug-in details panel.

To add plug-in configuration properties by editing the `server.xml` file, add a `pluginConfiguration` element to the file. The element supports the following attributes:

webServerPort — The port the web server uses to listen for requests. The default value is `80`.

webServerSecurePort	The port the web server uses to listen for secure requests. The default value is `443`.
ipv6Preferred	Whether to prefer ipv6. The default value is `false`.
pluginInstallRoot	The plug-in install root
sslCertLabel	SSL certificate label
sslKeyringLocation	SSL keyring location
sslStashFileLocation	SSL stash file location

The actual plug-in configuration file must be generated either by using a utility in the Liberty profile developer tools, or by running an **`mbean`** command using JMX. In the tools, generate the file by right-clicking the server and selecting **Utilities Generate** → **Web Server Plug-in**.

To generate the plug-in configuration using **`mbean`**, use a JMX console, such as `jconsole`, to connect to the server. Note that the localConnector-1.0 feature must be enabled on the server. From the console, invoke the **`defaultPluginConfig`** generation mbean.

8.2 Using the package utility

Because a Liberty profile server is lightweight, it can be packaged easily with applications in a compressed file. This package can be stored, distributed to colleagues, and used to deploy the application to a different location or to another system. It can even be embedded in the product distribution.

A Liberty profile image is an archive file that contains one or more types of resources of the Liberty profile environment. The types are dependent on the topology that is deployed. You can extract them manually or you can use an extraction tool to deploy the file to one or more systems. Alternatively, you can use the job manager in the WebSphere Application Server Network Deployment product to deploy these images.

The package will contain all of the binaries, server configuration, and applications necessary to distribute the server.

8.2.1 Packaging a Liberty profile server using developer tools

Use the following procedure to package a Liberty profile server from the developer tools:

1. Stop the server.
2. From the Servers view, right-click the server and select **Utilities** → **Package Server**.

3. Enter a file name for the archive.
4. Select whether you want to include all server content (including the server binaries) or only the server applications and configuration.
5. Click **Finish**.

8.2.2 Packaging a Liberty profile server from a command prompt

Use the following procedure to package a Liberty profile server from a command prompt:

1. Navigate to the Liberty profile server installation root directory.
2. Run the command **`bin/server package server_name`**.

The **`package`** command supports the following options:

--archive	The name of the output file
--include	Include the entire server configuration by specifying **`all`**. Include only the server applications and configuration by specifying **`usr`**.

8.2.3 Using the Job Manager to package and distribute Liberty profile servers

In WebSphere Application Server V8.5 Network Deployment, use the Job Manager to perform these functions:

- Package the Liberty profile runtime environments, configurations, and applications.
- Distribute and deploy a Liberty profile server and applications.
- Start embedded profile packages.

For more details about how to package the Liberty profile using the job manager, refer to the information center at the following website:

`http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/tagt_jobmgr_comp_server.html`

8.3 Moving an application to the full profile

Applications developed on the Liberty profile server can also be redeployed to run on a WebSphere full profile server. Note that the WebSphere full profile environment provides some features not available in the Liberty profile server.

Applications developed in the full profile environment might not run on Liberty profile.

8.3.1 Programming model differences between full profile and Liberty profile

The WebSphere full profile server and the Liberty profile server differ in programming support for the following technologies:

- Java EE 6
- Enterprise OSGi

For example, the full profile supports Java EE technologies such as Enterprise Java Beans (EJB) and the Java API for XML Based Web Services (JAX-WS), but Liberty profile does not. For OSGi applications, Liberty profile does not support Blueprint security.

For a more detailed comparison of technologies supported by the full profile and Liberty profile, refer to the following information center website:

`http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.express.doc/topics/rwlp_prog_model_support.html`

8.3.2 Configuration differences between full profile and Liberty profile

Although applications developed in the Liberty profile environment can be run on the full profile, there might be some configuration differences that need to be addressed.

General concerns

In the Liberty profile, many properties that represent time values can be specified using units of time. For example, one hour can be specified as the value `1h`. In full profile, time values are typically expressed as numeric values. When migrating application resources such as data sources or connection managers to full profile, you might need to modify property values to be specified as numbers.

Class loading

The Liberty profile server provides different methods of controlling class visibility than the full profile server. If your application requires you to configure advanced class loading behavior, you may need to reconfigure class loading in the full profile environment. You may also need to configure class loading in full profile if your application embeds classes that conflict with the full profile runtime. For

more information about class loading in full profile, see the following page in the InfoCenter:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.express.doc/ae/trun_classload.html

Data sources

Some data source properties have different names:

- `ifxIFX_LOCK_MODE_WAIT` is `informixLockModeWait` in the full profile.
- `supplementalJDBCTrace` is `supplementalTrace` in the full profile.

Some data source properties have different default values:

- begin`TranForResultSetScrollingAPIs` is true by default in the Liberty profile
- begin`TranForVendorAPIs` is true by default in the Liberty profile
- `statementCacheSize` is `10` by default in the Liberty profile

The Liberty profile allows `connectionSharing` to be configured to either `MatchOriginalRequest` or `MatchCurrentState`. By default, it is `MatchOriginalRequest`.

The full profile allows `connectionSharing` to be configured in a finer grained manner, where individual connection properties can be matched based on the original connection request or current state of the connection. In the full profile, `connectionSharing` is a combination of bits representing which connection properties to match based on the current state of the connection. In the full profile, a value of `0` means match all properties based on the original connection request, and a value of `-1` means to match all properties based on the current state of the connection. The default value for the full profile is `1`, which means that the isolation level is matched based on the current state of the connection and all other properties are matched based on the original connection request.

Connection Manager

Some of the connection manager properties have different names in the full profile and Liberty profile. Table 8-1 shows the property naming differences.

Table 8-1 Properties with different names

Liberty profile property name	Full profile property name
maxConnectionsPerThread	maxNumberofMCsAllowableInThread
maxIdleTime	unusedTimeout
maxPoolSize	maxConnections
minPoolSize	minConnections

There are also differences in the way timeout values for immediate or never (disabled) are specified. In Liberty profile, `0` represents an immediate timeout and `-1` represents a disabled timeout. In full profile, `-1` represents an immediate timeout and `0` represents a disabled timeout.

The value of the purge policy can also differ slightly between full profile and Liberty profile. There are three possible values in Liberty profile: `EntirePool`, `FailingConnectionOnly`, and `ValidateAllConnections`. The `EntirePool` option maps directly to the `EntirePool` option in full profile. `FailingConnectionOnly` maps to the `FailingConnectionOnly` option with the `defaultPretestOptimizationOverride` property set to `false` in the full profile. `ValidateAllConnections` corresponds to `FailingConnectionOnly` with the `defaultPretestOptimizationOverride` property set to `true` in full profile.

Security

In the Liberty profile, you can configure user-to-role mappings and runAs users in the `application-bnd` element of the `server.xml` file. In the full profile, you can only configure this in the `ibm-application-bnd.xml/xmi`.

Liberty profile has the following security limitations as compared to full profile:

- No Java EE security.
- Not all public APIs and SPIs are supported. The Java API document for each Liberty profile API is detailed in the Programming Interfaces (APIs) section of the information center, and is also available as a JAR file under the `/dev/ibm-api/javadoc` directory of the server image.
- No custom user registry.
- No horizontal propagation.
- No **`SecurityAdmin`** MBean support; therefore, methods such as clearing the authentication cache are not available.
- No Java Authorization Contract for Container (JACC) support.
- No Java 2 Connector (J2C) principal mapping modules support.
- No Java Authentication SPI (JASPI) support.
- No multiple security domain support.
- No security auditing subsystem that is part of the security infrastructure of the server.

Web applications

The Liberty profile server does not automatically expand web archive (WAR) files that are deployed to the server. The Java EE specification states that the

`getRealPath()` method returns a null value if the content is being made available from a WAR file.

If your application relies on a result being returned by `getRealPath()`, you must deploy the application as an expanded web application, not as a WAR file. For example, you can manually extract the WAR file and copy the expanded application to the dropins directory.

JSP

Full profile supports a useInMemory configuration option to only store translated JSP files in memory. Liberty profile does not support this option.

JAX-RS

The Liberty profile developer tools do not support third-party JAX-RS APIs. The developer tools for full profile include this support.

Bean validation

There is no support for bean validation inside OSGi applications in Liberty profile.

8.4 Using the Liberty profile on z/OS

WebSphere Application Server V8.5 provides features for administering a Liberty profile on a z/OS platform. You can use IBM MVS™ operator commands to start, stop, or modify the Liberty profile.

Additional optional features provide enhanced integration with z/OS qualities of service:

- Classify inbound HTTP requests with WLM
- Use a DB2® Type 2 driver with RRS transaction management
- Authenticate users using a SAF user registry
- Authorize users using a SAF authorization provider

The IBM Installation Manager is used to install the Liberty profile on z/OS using a part of the `com.ibm.websphere.zOS.v85` package offering.

For more information, refer to the following information center website:

http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.zseries.doc/topics/twlp_admin_zos.html

A

Additional material

This book refers to additional material that can be downloaded from the Internet as described in the following sections.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG248076`

Alternatively, you can go to the IBM Redbooks website at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG248076.

Using the Web material

The additional Web material that accompanies this book includes the following files:

File name	*Description*
ch3_ITSOWeb.zip	Zipped Liberty profile developer tools project with code samples from Chapter 3
ch4_OSGiExample.zip	Zipped Liberty profile developer tools project with code samples from Chapter 4
ch5_ITSOWeb.zip	Zipped Liberty profile developer tools project with code samples from Chapter 5
ITSO_derby.zip	Zipped Derby database for use with Chapter 5 samples
ITSO2_derby.zip	Zipped Derby database for use with Chapter 5 samples
ITSODD_derby.zip	Zipped Derby database for use with Chapter 5 samples

Downloading and extracting the Web material

Create a subdirectory (folder) on your workstation, and download the contents of the web material file into this folder. Extract the `sg248076.zip` file into this folder to access the sample application files.

Importing the zipped Liberty profile developer tools projects

To use any of the zipped Liberty profile developer tools projects, you must first create a Liberty profile server runtime environment in your eclipse installation as described in 2.2.1, “Installation using the Liberty profile developer tools” on page 26.

Use the following process to use any of the three example projects, `ch3_ITSOWeb.zip`, `ch4_OSGiExample.zip`, or `ch5_ITSOWeb.zip`:

1. Extract the zipped project file into its own subdirectory.
2. From eclipse, run **File** → **Import** → **Existing Projects Into Workspace**.
3. In the Import Projects panel, enter the path to the subdirectory where you extracted the zip file in the Select root directory field. This should fill the Projects field with either ITSOWeb (for the Chapter 3 and 5 examples), or five projects beginning with ITSO.OSGiExample (for the Chapter 4 example).
4. Select the box to copy projects into the workspace.
5. Click **Finish**.

Using the zipped Derby database files

To use the zipped Derby database files, simply extract the zip files and make a note of the file path. The examples in Chapter 5, “Data access in the Liberty profile” on page 87 describe how to set up a data source to point to these example databases.

Related resources

The resources listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

Online resources

These websites are relevant as further information sources:

- WebSphere Application Server Liberty Profile Information Center:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/twlp_inst.html

- Technologies supported by the Liberty profile:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.doc/topics/rwlp_prog_model_support.html

- Supported operating systems for the Liberty profile:

  http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg27028175

- Requirements for running the Liberty profile on z/OS:

  http://www-01.ibm.com/support/docview.wss?uid=swg27024798

- WASdev community:

  http://wasdev.net/

- WASdev forum:

  http://www.ibm.com/developerworks/forums/forum.jspa?forumID=2666

- Stack Overflow tags:

  http://stackoverflow.com/questions/tagged/websphere-liberty

- WebSphere Application Server Developer Tools for Eclipse:

  http://publib.boulder.ibm.com/infocenter/radhelp/v8r5/topic/com.ibm.rad.install.doc/topics/t_install_wdt.html

- OSGi home page:

  http://www.osgi.org

- Enterprise Bundle Archive (EBA) Maven plug-in:

  http://aries.apache.org/modules/ebamavenpluginproject.html

- Configuring database connectivity in the Liberty profile:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/twlp_dep_configuring_ds.html

- Application defined data sources:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/rwlp_ds_appdefined.html

- Liberty profile: How data source configurtion updates are applied:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/rwlp_ds_config_updates.html

- Securing the Liberty profile and applications:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/twlp_sec.html

- Authenticating users in the Liberty profile:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/topics/twlp_sec_authenticating.html

- Add logging to your applications:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/ttrb_javalogapps.html

- Monitoring the Liberty profile using MXBeans:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.zseries.doc/topics/twlp_mon.html

- How to package the Liberty profile using the job manager:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/tagt_jobmgr_comp_server.html

- Comparison of technologies supported by Full profile and Liberty profile:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.express.doc/topics/rwlp_prog_model_support.html

- Class loading in Full profile:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.express.doc/ae/trun_classload.html

- Administrating the Liberty profile on z/OS:

  http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.wlp.zseries.doc/topics/twlp_admin_zos.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM Redbooks

WebSphere Application Server Liberty Profile: Guide for Developers

(0.2”spine)
0.17”<->0.473”
90<->249 pages

WebSphere Application Server Liberty Profile Guide for Developers

Learn WebSphere Application Server profile designed for developers

Lightweight, easy to install, fast to use application server

Simplified development with the Liberty profile

WebSphere Application Server V8.5 includes a Liberty profile, which is a highly composable, dynamic application server profile. It is designed for two specific use cases: Developer with a smaller production runtime, and production environments. For a developer, it focuses on the tasks a developer does most frequently and makes it possible for the developer to complete those tasks as quickly and as simply as possible. For production environments, it provides a dynamic, small footprint runtime to be able to maximize system resources.

This IBM Redbooks publication provides you with information to effectively use the WebSphere Application Server V8.5 Liberty profile along with the WebSphere Application Server Developer Tools for Eclipse, for development and testing of web applications that do not require a full Java Platform. It provides a quick guide on getting started, providing a scenario-based approach to demonstrate the capabilities of the Liberty profile along with the developer tools, providing a simplified, but comprehensive, application development and testing environment.

The intended audience for this book is developers of web and OSGi applications who are familiar with web and OSGi application concepts.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-8076-00 ISBN 0738437468